DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
MAYNARD, MASSACHUSETTS 01754 / TEL. 897-5111 / TWX 710 347-0212

```
RRRRRRRRRR     SSSSSSSSSS     TTTTTTTTTT     SSSSSSSSSS
RRRRRRRRRR     SSSSSSSSSS     TTTTTTTTTT     SSSSSSSSSS
RRR    RRR     SSS               TTT         SSS
RRR    RRR     SSS               TTT         SSS
RRRRRRRRRR     SSSSSSSSSS        TTT         SSSSSSSSSS
RRRRRRRRRR     SSSSSSSSSS        TTT         SSSSSSSSSS
RRR    RRR             SSS       TTT                SSS
RRR    RRR             SSS       TTT                SSS
RRR    RRR     SSSSSSSSSS        TTT         SSSSSSSSSS
RRR    RRR     SSSSSSSSSS        TTT         SSSSSSSSSS
```

Dear RSTS attendee:

The RSTS US Special Interest Group has compiled this set of handouts for the RSTS related sessions at DECUS. The presenters, who have contributed information for this handout, deserve your appreciation for quickly putting handouts together, so that you would be able to attend their sessions with a knowledge of what will be discussed.

The handouts are divided into two sections: the User session and the DEC RSTS development group session handouts.

The task of planning the RSTS sessions has become increasingly more difficult. We have reached the point where we have a full four day schedule and are forced to overlap some sessions. Bill Sconce is the person responsible for the super scheduling job at this Symposium. Bill and I are actively looking for ways we can improve the meetings for you, so if you have any suggestions, please corner one of us during the meeting and let us know about it. At the back of this handout you will find a questionaire for the meeting and the RSTS SIG. Please take the time to fill it out and put it in the RSTS questionaire box or mail it to me. The only way we can improve the SIG is to have your ideas.

Bob Branton
RSTS US SIG CHAIRMAN

RSTS USER SESSIONS HANDOUTS

CONTENTS

Session	Room	Day	Time	Page
Structured Methodologies for BASIC PLUS and BASIC-PLUS-2 Programing	Poster Paper			1
Signed Number Arithmetic	Sante Fe	Mon	9:00	3
EPG Product Panel	Sante Fe	Mon	9:30	4
Programming Standards, Do they help the average timesharing user?		Mon	1:30	5
Meaningful and Manageable BASIC-PLUS Program Documentation	Golden West	Mon	2:00	15
System Access Management	Golden West	Mon	4:15	22
RPL11 - A Report generator for RSTS	Cabinet	Tue	8:30	25
Commercial Sub-SIG Meeting	Golden West	Tue	1:30	26
Establishing an Internal Time-sharing Service	Golden West	Tue	2:30	27
Educational Sub-SIG Meeting	Sen/Comm	Tue	3:15	28
Concepts for Systems Operations	Golden West	Tue	3:15	29
Major Ingredients of a General Accounting Package under RSTS	Golden West	Tue	3:45	31
A Thing called ZAP	Sen/Comm	Tue	4:15	38
MAPS - An Interactive Financial Modeling Language	Golden West	Tue	4:15	39
FORTRAN under RSTS	Golden West	Tue	4:45	40
System Security in an Educational Environment	Deanza/Mesa	Wed	8:30	41
On Line A/R Entry Under RSTS	Golden West	Wed	2:30	42

RSTS USER SESSIONS HANDOUTS

How to use RSTS/E: A User-Oriented Training Package	DeAnza/Mesa	Thur	10:15	51
User-Oriented Publications: On RSTS/E and for RSTS/E	DeAnza/Mesa	Thur	10:45	55
DYNPRI Design Considerations	Sunset	Thur	2:30	60

DIGITAL SESSIONS HANDOUTS

Datatrieve-11: DEC Query/Report System for PDP-11	Golden West	Thur	9:45	61
BASIC+2	Council/ Chamber	Tues	9:45	69
Overview of the RSTS System	Golden West	Mon	10:45	82

RSTS USER SESSIONS HANDOUTS

RSTS/E Accounting and Internal Structure	Senate/ Committee	Tues	4:45	91
RSTS/E WISHLIST	Golden West	Wed	10:45	92
RSTS/E New Users' Session	Golden West	Wed	3:15	97
RPL11	Golden West	Tues	8:30	98
QUESTIONNAIRE				117

POSTER PAPER

STRUCTURED METHODOLOGIES FOR BASIC-PLUS AND BASIC-PLUS-2 PROGRAMMERS

Presentor: Al Cini
Digital Software Services

As everyone knows, people spend a lot of time fixing, extending, and otherwise modifying existing programs. In fact, people spend a little more time on the average maintaining software than they do developing it. Nationally, almost 60 cents of every software manpower dollar is diverted, for one reason or another, into maintenance efforts. Techniques which might reclaim even a portion of this cost, therefore, promise significant long-term cash benefits to anyone with money invested in custom software (whether developed/maintained in-house or vendor-supplied).

Such solutions so far have centered on the imposition of software standards. Quite simply, software standards are sets of rules which programmers and other software personnel follow in developing and maintaining software. Adherence to common work habits, it is felt, will lead to interchangeable, maintainable, and inherently more reliable programs. Standards are often installation-specific, usually arbitrary, and almost never popular. While standards which govern the documentation level of software activity are most common, standards may be set at the programming and design levels as well.

At the documentation level, standards will govern the format of programming language statements, selection of variable names, the format and frequency of comments, and other factors which affect the cosmetic characteristics of a program. Intended to improve the legibility of programs, documentation standards promote coding uniformity and, incidentally, may improve maintainability. The widely distributed BASIC-PLUS SOFTWARE CONVENTIONS document, which outlines the specifications that every BASIC-PLUS and BASIC-PLUS-2 program written by DEC personnel must follow, is a classic example of documentation standards applied specifically to BASIC-PLUS software.

At the programming level, standards will specify the kinds of control structures acceptable in an application program written for a specific installation. The ACCEPT statement may be outlawed for COBOL programs, the FORTRAN ASSIGN'd GOTO may be blacklisted, transfers into and out of loops may be restricted, and other language tools related to the specific machine implementation of data structures and algorithms may be controlled. Cohesive standards of general applicability are difficult to find at this level, but installation specific guidelines aimed at specific languages and applications abound. Structured programming, widely referred to as "GOTO-less" programming, is an example of a programming standard.

Structured Methodologies for
BASIC-PLUS and BASIC-PLUS-2 Programmers

Design level standards outline acceptable procedures for resolving real-world requirements into functional specifications detailed enough to be assigned to programmers. Formal guidelines are rarely found at this level. More commonly, software designers will proceed through systems analysis with some intuitive attention paid to informal and almost ritualistic "standard" activities, such as, forms or card layouts and PERT charts.

This is the "Inside-Out" design philosophy, which often fails to produce a plan specific enough for a programmer to follow. More formal design standards include the "Bottom-Up" and "Top-Down" strategies, the latter of which along with structured programming, is generally considered a "structured methodology."

In the late 1960's, Böhm and Jacopini demonstrated that all flow-chartable programs can be written using combinations of only three control structures:

Sequence: simply, this means that one statement is executed after another.

Selection: IF-THEN-ELSE structure, which selects alternate and mutually exclusive paths through a program based on the true/false evaluation of one or more conditions.

Iteration: a section of a program can be repeated until (or while) a condition is true.

Soon after, computer scientist Edsger Dijkstra proposed that we take a "constructive" approach to the development of software such as to produce "a priori correct programs." Structured programming evolved out of this early, theoretical work, and is still a source of debate among data processing professionals.

The DECUS presentation will expand on the basic structures, with emphasis on rules for converting them to BASIC structures using BASIC-PLUS and BASIC-PLUS-2. At the same time, techniques for using top-down design to arrive at structured programs, and documentation considerations specific to structured code, will be discussed. Structured programming as a technique for improved programmer training curricula, and the results of some structured programming experiments in actual projects will be treated as well.

SIGNED NUMBER ARITHMETIC

Presentor: John A. Christensen
Idaho Falls School District

Day: Monday, Nov. 28
Time: 9:00-9:30
Room: Santa Fe

The signed number arithmetic package is a set of programs written in BASIC-PLUS designed to both teach students about signed numbers and drill them on signed number facts. The tutorial package teaches students how to add, subtract, multiply and divide signed numbers. Signed numbers in this package refers only to signed integers.

The tutorial package begins by intoducing students to the postman story. The postman story is a neat real life story to aid students in understanding signed numbers and their operations. This story will be discussed in detail. From the story the student graduates to learning signed number arithmetic facts in order to perform the necessary operations. The student must score satisfactorily on each program before he is allowed to progress to the next program, which teaches another concept. This package consists of 8 programs.

The drill package is 7 programs designed to give a student who is already familiar with signed number facts practice doing signed number operations. The programs are competive and can be used in competing with other students or in attempting to improve a students score.

In this session we will illustrate this complete package which should be a source of both materials and ideas. . . .

EPG PRODUCT PANEL

Presenter: Representatives from Education Products Group

Day: Monday, November 28
Time: 9:30 a.m. - 11:45 a.m.
Room: Santa Fe Room

Representatives from the Education Products Group will be available for a two-part panel discussion. Primary emphasis will be placed on an Input-Response session whereby EPG will formally respond to the concerns of the educational community. We will work with EDUSIG to obtain a list of items of concern well in advance of this panel so that reasonable responses may be prepared.

An outline of the format of this Panel follows:

45 minutes - A structured presentation of EPG products; specifically, enhancements to existing products.

45 minutes - Input/Response and Question and Answer session.

EPG will also be displaying a booth in the Exhibit Hall where we plan to demonstrate our Mark Sense Batch -11 system. Please stop by our booth and give us an opportunity to meet you.

PROGRAMMING STANDARDS, DO THEY HELP THE AVERAGE TIMESHARING USER?

Steven Stepanek

California State University, Northridge

Monday Nov 28 1:30-2:00 pm

INTRODUCTION

To the programmer:

The Library Standards were created to make the use of a library program as easy and as trouble free as possible for the USER. Additionally, having Library Program Standards assure the user that he is using a reasonably well-debugged program; one that can be used easily and one that can be used with confidence that results are accurate. As a programmer, you should realize that these standards were not implemented to interfere with your programming style or to inhibit your creativity. Therefore, reasonable exceptions to these proposed guidelines may be accepted. However, programs will be evaluated against these Standards, and preference will be given to programs that provide ease of use, proper documentation, and allow ease of maintenance.

If you wish to submit a program for inclusion in the Library, you should "imagine" that:

1) the user is utilizing the program to supplement his knowledge of the programmed application
2) the user has no programming knowledge.

This will not always be the case, but by assuming minimum programming knowledge on the part of the user and by orienting your instructions toward the unsophisticated user, you will provide a program which is easy to use successfully.

I. JUSTIFICATION FOR INCLUSION IN THE TIMESHARING LIBRARY

A. USAGE

The program should be of use and interest to a reasonably large group of users. It is preferable that the program be useful to more than one class, to more than one campus, and for more than one school term.

If the program is written for a specialized group of users, such as electrical engineers, geographers or economists, the programmer should provide reasonably non-technical information on the program's use.

If the program is currently in use, the program owner should provide an estimate of utilization. For instance, the submitter should provide information regarding the number of students using the program as well as the number of classes using it. If more than one campus is using the program, this information should also be included.

B. UNIQUENESS

The program should be unique, that is, it should contribute something that existing library programs do not accomplish. These unique features should be clearly stated in the self-contained documentation of the program.

C. STANDARDS

Lastly, the program submitted must meet the Library Program Standards to a reasonable degree as determined by the system Librarian before the program will be considered for installation on the system.

II. DOCUMENTATION

A. PROGRAM NAME

1. A library program name has the same restrictions as a filename on the system.

2. The name should be mnemonic ,i.e., "sound" like what the program "does".

   EXAMPLE: MATINV, INVERS or MINVRS for a program to calculate the inverse of a matrix with a name limited to six characters.

3. The program name should be printed when the program begins execution. The characters in the name should be

spaced out - a blank between each character to improve readability.

4. A blank line and a brief one-line title or description should be printed below the program name. The program name and one-line description constitute the program heading.

   EXAMPLE: M A T I N V

   CALCULATION OF THE INVERSE OF A MATRIX

   In some cases where a program is specialized or is one of a series of programs a third line may be added to the heading to indicate this.

   EXAMPLE: A NUTRITION ORIENTED PROGRAM
   or
   A HUNTINGTON II PROGRAM

5. The heading should be preceeded by 2 blank lines. Two blank lines should follow the heading.

6. The program name may have to be altered due to certain system restrictions. When possible, a name change will be discussed with the program's submitter.

B. PROGRAM DESCRIPTION

A reasonable and short abstract of the program should be provided for inclusion in published as well as on-line indices of library programs available.

C. SELF-CONTAINED PROGRAM INSTRUCTIONS

1. After the program has begun execution and the program heading has been printed, the user should be given the option to obtain instructions and/or a program description which will aid him in using the program. The query:

   DO YOU WANT INSTRUCTIONS?

   should appear. If the user responds "YES", one line should be skipped and the instructions or description of the program should be printed. If the user responds "NO", the program should omit the printing of the self-contained instructions, skip a line, and continue with the program.

   Self-contained instructions should be clear, concise, and printed in less than 50 lines. Additionally, they should be understandable to as large a group of users as

possible. The instructions should be explicit, clearly written, and explain what the program does. The instructions should also give the limitations of the program, and explain how to use the program.

2. The instructions should specify the manner in which data is to be entered. In statistical applications the PREFERRED method is to enter data by case or observation.

   EXAMPLE: If there are 10 cases and 3 variables (e.g., age, number of cigarettes smoked per day, and number of visits to the doctor last year) all the data points (i.e., variables) for the first case should be entered before entering the variables for the second case, etc.

3. Where necessary, the program's self-contained instructions should provide information regarding the method of calculations, and program peculiarities. In all cases program limitations should be included in the self-contained instructions, i.e., maximum number of variables and maximum number of cases, etc.

4. A maximum terminal width of 72 characters should be assumed. All program output should use as much of the print line as possible. This conserves paper and compacts output.

5. Left justify the beginning of paragraphs or when asking for input. Blank lines should separate paragraphs. Likewise a blank line should separate requests for input.

6. Continuing with the example of the hypothetical program MATINV, a typical description could look like the following (user responses underlined):

   M A T I N V

   MATRIX INVERSION BY GAUSS-JORDON ELIMINATION

   DO YOU WANT INSTRUCTIONS? YES

   MATINV COMPUTES THE INVERSE OF A MATRIX WITH 2 TO 30 ROWS AND COLUMNS BY GAUSS-JORDON ELIMINATION. THE USER MUST INPUT THE ORDER OF THE MATRIX (I.E., NUMBER OF ROWS/COLUMNS) AND THE MATRIX ITSELF.

   THE MATRIX MAY BE ENTERED FROM THE TERMINAL OR A FILE.

THE OUTPUT, CONSISTING OF THE INVERSE OF THE MATRIX, MAY BE PRINTED ON THE TERMINAL OR WRITTEN TO A USER-NAMED FILE. THE DETERMINANT IS PRINTED OUT AND THE RANK IS GIVEN WHEN THE MATRIX IS NOT INVERTIBLE.

7. When instructions are lengthy or complicated it helps to have detailed user "prompts" distributed where necessary in the program.

D. SEPARATE DOCUMENTATION

1. Separate documentation is desirable when instructions are too long to be printed at the terminal. Separate documentation should be clearly written, concise, state example problems, and show sample executions of these problems.

2. Separate documentation may also be required because the program is very specialized, such as a continuous system simulator or electronic circuit analysis program. The instructions should be as nearly self-contained as possible. Where necessary, references to EASILY AVAILABLE supplemental materials should be mentioned in the separate doucmentation.

3. If the program requires separate documentation either because the instructions are too lengthy to be printed at the terminal or because the program is specialized, the program should notify the user that separate documentation exists. After the program heading is typed, the program should print:

SEPARATE DOCUMENTATION IS REQUIRED FOR THIS PROGRAM.

DO YOU WISH TO CONTINUE?

The availability of separate documentation will be contained in publications concerning Timesharing Library Programs.

4. Users submitting separate documentation should send a photo-ready copy to the Timesharing Librarian. Any separate documentation should be typewritten.

If the program is accepted for the Timesharing Library (and commercially published manuals or instructions are not available), the separate documentation will then be made available IF POSSIBLE.

If a program will require separate documentation, it is advised that the submitter ascertain the availability of published materials or the printing thereof before a program is produced or submitted for inclusion in the

library. Should published program accompaniments not be available, it would not normally be desirable to place the program in the library.

E. SPECIFICATIONS

Specifications consist of supplementary information for programs with self-contained instructions. This supplementary information is not essential to the running of the program, and is provided to the user for additional insight to the program's operation. Specifications include such items as the mathematical equations used, information about the algorithms used in writing the programs, non-standard assumptions, etc. References should be ennumerated in case the user would like to pursue related information in depth.

F. SAMPLE DATA

1. Sample data should be included when submitting a program for the library. If possible, this data should be taken from a standard text which can be referenced easily in case additional testing of the program is desired. The correct "solution" to the sample data used in the program should be cited in such a reference. Sample data should be sufficient to demonstrate the prime features of the program and to thoroughly test the program.

2. Small sets of sample data are generally presented by a sample execution of the program where the data is entered via the terminal.

3. If the sample data is on a file, the file should be supplied in machine readable format or made available for public access when the program is submitted. A listing of the sample data should also be included with the sample execution of the program showing data entered from a file.

III. PROGRAMMING CONVENTIONS

A. COMMENTS AND REMARKS

Internal or "programmer" documentation should be explicit and complete. There should be no question as to what is "happening" in any major segment of the code.

B. PROGRAM ID

Each program should begin with a comment section and include at least the following information:

Program name
Short program description (no more than 5 lines)
Program author or the person who converted the program
Date program was written or converted
Author's department and campus address

C. QUESTION TO THE USER

1. SYNTAX

Requests for responses from the user should be unambiguous. In a question where the answer is obviously "YES" or "NO", the "YES" or "NO" should NOT be included in the request for a user response. Otherwise it is preferred that the acceptable responses be specified.

A good question is:

DO YOU WANT TO RUN ANOTHER PROBLEM?

A bad example is:

TYPE "1" TO CONTINUE.

In the bad example, the user might type quotes around the 1. Additionally, he is not told what to do if he doesn't want to continue. And while the response is easiest to think of in terms of yes or no, it requires a numeric answer.

2. FORM

When requesting a response, follow the request by either a question mark (?) or a colon (:) AND A SPACE. The space is necessary to provide readability after the user responds.

EXAMPLE: HOW MANY SETS?
or
ENTER UPPER BOUND:

3. CHECKING THE USER'S RESPONSES

The answers to ALL questions should be checked for legality. If the user enters an invalid response, the program should prompt him to re-enter the proper answer to the question.

EXAMPLE: (user responses are underlined)

HOW MANY CASES? -10

If the maximum number of cases that a program can handle is 200 the limits should be restated when an invalid response is given...

NUMBER OF CASES MUST BE BETWEEN 1 AND 200:

4. BREVITY

Don't ask the user to type in lengthy answers if possible. The exception is the lengthy response necessary to reinforce concepts presented in the program. Use of such responses as "conditioning" and "reinforcement" is preferable in a behavioral modification program.

When possible suggest a short response:

ARE YOU BEGINNING OR CONTINUING (B OR C)?

which will invite less typing errors than:

ARE YOU BEGINNING OR CONTINUING?

F. DATA ENTRY

1. DATA FROM TERMINAL OR FILE

The user should have the option of entering his data from the terminal or a user-named file:

WILL YOU ENTER DATA FROM A FILE OR TERMINAL?

If the data is to be entered from a file, the next question the program should ask the user is:

NAME OF INPUT FILE?

NOTE: IF THE DATA IS TO BE ENTERED FROM THE TERMINAL, THE PROGRAM SHOULD NOTIFY THE USER THAT IT IS WAITING FOR DATA.

EXAMPLE: (user responses are underlined)

READY FOR INPUT DATA.
EACH ROW REPRESENTS ONE CASE.
THERE SHOULD BE 4 VALUES FOR EACH ROW.

ROW 1: 10.0,15.0,20.0,30.0
ROW 2: 20.0,15.0,52.0,30.0
ROW 3: 1.0,15.0,82.0,30.0

In this example, the user has previously responded that he has 3 cases with 4 variables. The program thus

carefully reminds the user that he will have 4 entries per row, then the program types out the row number and waits for the user to enter the data for that row.

2. MINIMAL DATA

The requirement of giving the user the option of having his data on a file may be dispensed in cases where the data input to a program is expected to be minimal.

3. PARAMETERS

Programs must be designed to accept only actual data from files. Files should NEVER be expected to contain program parameters, such as the number of cases and the number of variables. These parameters, or information concerning the data should be entered from the terminal as part of the user's interaction with the program.

G. FILE HANDLING

If your program creates a scratch file in the user's account that is no longer needed after the execution of the program, include the code to delete the file before terminating the program. The user should also be notified that the program creates a scratch file.

H. INPUT/OUTPUT STATEMENTS

The use of variable logical file numbers (or channels) is a convenient way to handle input/output. Depending upon the user's response, the input statement can accept information from the terminal or a file. In either case, the use of variable logical file numbers have the advantage of being flexible and being more easily converted to other systems.

I. TERMINAL INTERRUPT

In answering program questions or in entering data, the user may enter the incorrect response. When a user changes his mind or has made an error he should be able to initiate a terminal interrupt to terminate the current processing operation of the program. The program should be coded so that it attempts to "send" the user to an appropriate "place" in the program. Normally, a terminal interrupt during the execution of a program will terminate the program and return control to the subsystem. This means, of course, the user must reload the program and begin again. The object of programming the interrupt is to save the user's time and to reduce his frustration.

J. OUTPUT OPTION

If the program can produce large amounts of printed output the user should have the option of routing the output to a user-named file for later printing (either to the terminal or to the system printer).

K. PROGRAM END

Two blank lines should follow the last printed line of the program.

V. PROGRAM STRUCTURE SUGGESTIONS

Try to keep statement numbers in ascending order. This will make the program easier to follow on GO TO statements and other transfer of control statements.

Before submitting a program resequence the line numbers in increments of 10.

Keep the program modular, e.g., rather than create one huge main program requiring a long compile time, break the program into smaller and easier to debug subroutines.

Within the constraints of the language choose variable names so that they are as mnemonic as possible. A variable dictionary in the program or in separate documention is desired.

Define constants used in many places throughout the program in variables at the beginning, so if the values of the constants need to be altered, only the lines at the beginning of the program need be changed to change all the values in the program.

These are not requirements for inclusion - but simply good programming techniques which will make the program easier to maintain.

VI. TESTING PROGRAMS

Before a program is submitted it should be run against the following procedures to see how well it can handle boundary conditions and user errors. These are the types of procedures that a program will be subjected to while being evaluated for the library. If you do submit a "test" run mark or notate your output as to the kind of testing being done.

Remember, with small to medium timesharing programs, as well as many large programs, ALL OPTIONS can be tested

and checked for validity.

1. BOUNDARY CONDITIONS

Test extreme values on input of zero, negative and positive numbers (both very large and small). Ask yourself if these values are reasonable, and if so do they produce reasonable results. Does the program do limit checking on input?

2. INTERRUPT TESTING

Test the interrupt processing one 'section' at a time. Wherever input is expected, hit the appropriate system interrupt key(s) -- then terminate the program. The next time through, continue the program. Often times you will find values have not been reinitialized properly. For interrupt testing always start program execution at the top of the program each time.

3. PATH TESTING

Test your program through all possible paths. When possible, change only one thing at a time so you'll know where to start looking for problems.

4. ANOTHER USER

Let someone else try your program. Did they understand your description or instructions? Did they know how to answer requests for input? Were there areas they weren't sure of what was happening? Did you ask them for a critique?

5. IDIOT PROOFING

Did you do everything possible to make sure your program is idiot-proof? Did you make the common mistakes new users make? Did you pretend you weren't interested or sleepy? Did the program help you along?

6. LAST MINUTE CHECK

In order to aid in the consistency of timesharing programs the following items will be carefully scrutinized:

HEADER:

3 blank lines
N A M E (characters separated by 1 space)
1 blank line
package name (if any)
1 blank line
description
2 blank lines

PUNCTUATION: It is expected that sentences be properly punctuated. Add one space after any punctuation.

INPUT: Whenever you ask the user a question, follow the question with a question mark (?) AND A SPACE. Whenever you are waiting for input, output an appropriate message followed by a colon (:) AND A SPACE.

PROGRAM END: Are there two blank lines following the last line your program prints?

VI. CONCLUDING REMARKS

These standards have been written to apply to any timesharing system. For this reason system specific items such as where and who to submit the program to and programming techniques specific to the system's languages have been attached as appendices. It is hoped that the body of this document can serve as guidelines for any timesharing system. The appendices can then be tailored to custom fit site or system pecularities.

VII. PROGRAM SUBMISSION

To submit a program for inclusion in the Timesharing Library, the user should provide the following:

A. PROGRAM LOCATION

The account number where the program is located. The user should NOT include account passwords or non-printing characters for his account.

B. SOURCE PROGRAM

The complete filename where the source program can be found. Binary or object files are not acceptable. If the filename contains security characters, such as non-printing control characters, the user should rename the file without security characters or any sort of attached password mechanisms.

C. DATA FILES

The complete filename of any sample data files (if used by the program). A reference to a standard source in which the data and/or results can be found should likewise be included. This is to aid in the verfication of the program.

D. ACCESS

ALL PROGRAM FILES and any data files should be defined for public access. To evaluate the program the librarian will need run, copy, and list capability.

E. LISTINGS

The source listing of the program and listings of any data file (if used by the program). Please make sure the source listing matches the sample execution that is submitted at the same time.

F. SAMPLE RUN

A COMPLETE sample run should be submitted. It is to show: execution with and without the self-contained instructions, interrupt handling at all levels, boundary testing and a systematic testing of all options. This sample run should especially demonstrate the most commonly used or unique features of the program. In all cases the sample execution should be notated, i.e. written comments should state what is being tested.

G. ADDITIONAL DOCUMENTATION

If the program requires additional separate documentation, a clean, photo-ready copy should be included.

H. WHO

All of this material should then be submitted to the Timesharing Librarian for the specific system.

I. RSTS/E LIBRARIAN

Questions about proposed programs for the library should be directed to the RSTS LIBRARIAN. Likewise programs should be submitted to the following person:

RSTS/E Librarian
Instructional Support Group
Division of Information Systems
The California State University and Colleges
5670 Wilshire Blvd.
Los Angeles, California 90036

II. SYSTEM SPECIFIC CODING SUGGESTIONS FOR RSTS/E BASIC-PLUS

The main body of the Timesharing Library Standards was kept system and site independent. This part of the appendix is meant to provide you with sections of coding or system specific references that will assist your programming on the RSTS/E system.

When appropriate, coding examples are given which correlate with many of the topics found in the main document.

Because of limitations imposed on program size by Basic-Plus, programmers should be aware of coding conventions that reduce program size and increase efficiency. See the RSTS/E Version 6A Programming Manual, Chapter 8, or the Basic-Plus Instant for a summary of techniques.

The most significant efficiency increase can be achieved by writing multiple statements per line number, using integer variables where possible, and putting comments at the end of a line using the exclamation mark (!).

NOTE: Care must be exercised in writing multiple statements per line number to maintain program readability and debugging ease. Please examine the following examples carefully for techniques to improve readability.

A. COMMENTS AND REMARKS

The liberal use of REM or ! (exclamation mark) in Basic-Plus is encouraged. In fact, a good rule would be: "IF THE READER COULD HAVE DOUBTS, COMMENT". Remember - the coder is almost always sure what he's doing; it's when the next person comes along and tries to figure out what is going on, i.e., comment for the next person. The preferred method is to use the exclamation mark (!) at the end of a line followed by a

comment. Readability is aided if all comments are lined up, i.e., start all comments in the same column.

```
        .
        .
100     C=P*D
        :A=P*R**2               !CIRCUMFERENCE AND AREA
                                !AREA OF CIRCLE
        .
        .
```

B. PROGRAM ID

Each program should begin with a comment section and include at least the following:

Program Name
Short program description (no more than 5 lines)
Date program was written or converted
Program author or the person who converted program
Authors department and campus address

C. YES/NO USER RESPONSE CODING

User's responses should be checked for both YES and NO. Don't assume that if there isn't a match on YES that the user typed NO. He may not know what is required. The program should allow a user to specify his answer either by Y or YES or, N or NO. The program should also take into account that the input may be in lower case characters. The CVT$$ function is used to change lower case to upper case characters, as shown below.

Because string handling is slow, the comparisons for the response is done with integers to increase speed and save core needed for the code.

NOTE: A prompt for the proper response is given in the event of an incorrect response. The original question should not be repeated when the prompt is given.

Using a GOSUB to decrease program size:

```
        .
        .
100     PRINT "DO YOU WANT INSTRUCTIONS";:
        GOSUB 20000:
        IF A%
           THEN 110
           ELSE 200             !INSTRUCTIONS?, YES
                                !OR NO
110!
!       BEGINNING OF INSTRUCTION BLOCK
!
        .
        .
200!
!       BEGIN PROGRAM EXECUTION
!
        .
        .
20000 A% = -1%                  !YES/NO GOSUB
20010 INPUT LINE X$
      :X% = ASCII(CVT$$(X$,39%))
      IF X% = 89%
         THEN RETURN
         ELSE IF X% = 78%
                 THEN A% = 0%
                      :RETURN   !RETURN WITH RESPONSE
20020 PRINT "YES OR NO";
      :GO TO 20010              !END YES/NO/ GOSUB
        .
        .
```



```
        .
        .
200!
!       BEGIN PROGRAM EXECUTION
!
        .
        .
20000 A% = -1%                  !YES/NO GOSUB
20010 INPUT LINE X$
      :X% = ASCII(CVT$$(X$,39%))
      IF X% = 89%
         THEN RETURN
         ELSE IF X% = 78%
                 THEN A% = 0%
                      :RETURN   !RETURN WITH RESPONSE
20020 PRINT "YES OR NO";
      :GO TO 20010              !END YES/NO/ GOSUB
        .
        .
```

Using a FUNCTION to decrease program size:

```
        .
        .
100     PRINT "DO YOU WANT INSTRUCTIONS";
        :IF FNA%
            THEN 110
            ELSE 200
110!
!       BEGINNING OF INSTRUCTION BLOCK
!
        .
        .
200!
!       BEGIN PROGRAM EXECUTION
!
        .
        .
20000 DEF FNA%
      :FNA% = -1%               !YES/NO FUNCTION
20010 INPUT LINE X$
      :X% = ASCII(CVT$$(X$,39%))
      :IF X% = 89%
          THEN 20030
          ELSE IF X% = 78%
                  THEN FNA% = 0%
                       :GO TO 20030
                                !DETERMINE ANSWER
20020 PRINT "YES OR NO";
      :GO TO 20010
```

```
20030 PRINT                          !YES/NO END
      .
      .
```

NOTE: The semicolon (;) in Basic-Plus suppresses the line feed after output. The proper punctuation for the questions are provided by the INPUT statement. It prints a question mark (?) followed by a space and waits for the user's response.

D. FILE HANDLING

Open files at the beginning of programs.....

A unique filename must be created for each user running a program because multiple users may be in the same account. The way to achieve this is to use the name of the program as the filename, and the user's job number as the extension. The following code will form the job number as a two character string ready to be used in a file open.

```
      .
      .
100   E$ = MID(NUM$((SWAP%(CVT%$(LEFT(SYS(CHR$(6%)
         +CHR$(9%)+CHR$(0%)),1%)))/2%)+100%),3%,2%)
                                     !JOB # AS 2 CHARS
                                     !FROM ERROR 0
      .
      .
```

NOTE: The user should be informed, in the program instructions that scratch file(s) are being created. The program should also delete the scratch file(s) when finished. This can be done by executing a KILL or UNSAVE statement in the program with the same string that was used to open the file.

F. PROGRAM INTERRUPT

Programmers should review the ON ERROR GO TO statement in the Basic-Plus language manual, page 8-5.

To enable program handling of a terminal interrupt (CONTROL-C) a SYS system function must be executed. The input and output buffers of the terminal are cleared, and program execution transfers to the line number specified in the last ON ERROR GO TO statement executed.

The SYS call to set the interrupt should be the first statement executed when handling interrupts, as the interrupt clears program interrupt control. As an example:


```
      .
      .
100   ON ERROR GO TO 20000
      :X$ = SYS(CHR$(6%)+CHR$(-7%))
                                     !SET ERROR CONTROL,
                                     !ENABLE CONTROL-C
      .
      .
5000  PRINT
      :PRINT "RUN AGAIN";            !TERMINATION OF
                                     !PROGRAM?
      .
      .
20000 X$ = SYS(CHR$(6%)+CHR$(-7%))
      :E% = ERR
      :F1% = LINE
      :RESUME 20010                  !RESET CONTROL-C,
                                     !SAVE ERROR CONDITIONS
                                     !CLEAR ERROR FLAGS
20010 IF E% = 28%
         THEN 5000                   !HANDLE CONTROL-C
      .
      .
```

NOTE: The RESUME statement must be used in conjunction with the ON ERROR GO TO statement. Line 20000 in the above example uses the RESUME to clear any error conditions before entering the error handling code. This prevents invertently not using a RESUME in the proper location.

II. SUMMARY OF CODE OPTIMIZATION TECHNIQUES

1. Write multiple statements per line.

2. Specify INTEGERS or REAL numbers using "%" or ".".

3. Use ! to precede comments rather than REM

4. IF statements; it is unnecessary to compare INTEGER values to zero. That is;

```
IF A% = 0%
   THEN 100
```

can be more efficiently written as

```
IF A%
   THEN 200
```

since any non-zero value is "TRUE".

5. Use statement modifiers whenever possible, i.e., IF, UNLESS, FOR, WHILE, and UNTIL.

6. Avoid Matrix commands whenever possible, i.e., MAT INPUT.

7. ARRAYS

   a. General index calculations are time consuming. Use individual variables whenever possible.

   b. Always dimension arrays; default dimensions may waste core. Also execute DIM statements early in a program's execution.

   c. Always insure array elements are integer when possible.

8. Re-use previously calculated items, avoid unnecessary intermediate calculations.

9. Avoid user defined functions, especially with string variables, if you can do the same thing with a subroutine.

10. If you require rounding for variables to be printed let the PRINT USING statement do it for you.

11. ALWAYS exit from subroutines using RETURN.

12. ALWAYS exit from ON ERROR GO TO coding using RESUME.

13. Avoid OPENing files in the middle of a program.

    Whenever a file is opened during program execution, buffer space in the user's program area is claimed. This is done most efficiently at the beginning of program execution when free contiguous space is usually more available. If files are opened in the middle of program execution and if sufficient contiguous space is not available at that moment, then the Garbage Collector will be called to re-shuffle the available space. The overhead of garbage collection can therefore be largely avoided if all files which the program will use are OPENed as soon as possible. For the same reason, re-opening a file should be avoided unless absolutely necessary.

14. Avoid dynamic string assignments and manipulations as much as possible.

    Consider four types of statements:


```
10  A$ = "ABCD"
    :B$ = SPACE$(5%)
20  C$ = A$
30  D$ = A$ + "E"
    :E$ = LEFT(A$,2%)
40  LSET B$ = A$
```

Statements like line 10 assign string variables and cause string space to be allocated in the user's area of core. Statements like line 20 are re-definitions of existing string data; they do not cause new space to be allocated but simply point the left-hand variable to the existing string data to which the right-hand variable is already pointing. Statements like line 30 are string-manipulations; they differ from the preceding type in that a new string is created, for which space must be allocated. Finally, statements of the type in line 40 are core-to-core data transfers; they do not cause any new space to be allocated but cause the string data to which the right-hand variable points to be copied into the string space (already allocated) to which the left-hand variable has been and is pointing.

Because statements such as those in line 10 and 30 must claim new string space and consequently may result in calls to the Garbage Collector, their use in a program can entail high run-time overhead. Consequently where execution speed is of concern, their use should be restricted to the minimum necessary. To this end the following guidelines will be helpful:

a. Assign at the beginning of the program all string variables which will be used. Thereafter move string data, when possible, by the LSET and RSET verbs (which can null-fill or truncate) rather than manipulate the strings.

b. Where string manipulations seem necessary, integer conversions (ASCII( ), CHANGE) and integer manipulations can often be substituted with less run-time overhead.

c. Where actual string manipulations seem unavoidable, do not use intermediate variables or more than one statement if not necessary.

MEANINGFUL AND MANAGEABLE

BASIC PLUS PROGRAM DOCUMENTATION

Presentor: Eric E. Beaverstock
Intelligent Systems Corp.
460 Totten Pond Road
Waltham, Mass. 02154

day: Monday Nov 28th, 1977
time: 2:00 PM - 2:30 PM
room: Golden West

Program documentation is a major issue for anyone trying to maintain code, run a program, or understand the logical flow of a system. This presentation shows an approach that provides a method by which program documentation is made meaningful and manageable and that implementation of a software system can be done in an orderly manner.

All stages of software development come under the control of a software library facility. When such a library and support software is implemented on a system, the user is able to organize, document, maintain, and control software development.

The library facility supervises the development of application software from the early design stages to the maintenance of operational documents. The following areas are covered.

The assignment of programs by the project director

Program creation and documentation

Program maintenance

The creation and maintenance of system operational documentation

The steps and procedures described are not only for the RSTS/E Basic Plus users, but may be used by anyone interested in having more control over any type of software development. This system is currently being used at Intelligent Systems Corp. The data presented results from experiences acquired through the implementation of the package and through the use of such a package to supervise program development.

I. Overview

During the development of application programs and maintenance of a documented system certain problems are bound to arise such as:

> What are the program specifications, and who is responsible for particular programs.
>
> Will all of the programs developed conform to a desired set of programming standards.
>
> What is the status of programs assigned to a particular application package.
>
> Are programs being written without being properly documented.
>
> Once bugs are found how does one control the assignment of programs and insure that the problems are solved.
>
> What can be done to guard against the situation where more than one person is working on the same program.
>
> How can one be assured that the final system operation manuals really do correspond to how the programs operate.
>
> Will program documentation changes find they way into a new manual.

Many of these types of problems can be solved through the implementation of a software library facility. All programs become part of the software library. The library is under the control of the software library facility. This facility is used to organize, document, and maintain the software library.

The softwary library facility controls the various stages of program development. When a program's function is defined, a program specification is created and entered into a computer file. The program is then "assigned" to a developer (programmer) through the library facility. As programs are developed, progress is posted as "percentage complete". Various reports show the current status of these assigned programs.

When the program is finished, it is "submitted" to the library through the library facility. The programmer notifies the librarian that the work is complete, and then the librarian verifies the work and performs the actual submission into the library.

Part of the library facility provides a mechanism of reporting, listing, and assigning program "bugs". When a bug is discovered, the user reports the problem directly to the library facility. The bug file is periodically reviewed by a manager who then can use the facility to make maintenance assignments.

The library facility controls the maintenance of programs as the programmer must first "withdraw" the program and/or related documents

from the library. After work is done on a program, the program must be resubmitted. Information about the withdrawls and resubmissions after maintenance are retained as a "maintenance history".

Since the program documentation standards are controlled by the library facility, an automatic procedure can be used to aid in the creation of user manuals, system manuals, and other such documents.

II. The library

A. Design

The "software library" is a collection of disk files. These files consist of the following:

Source code
Program (capsule) documents
Program specification documents
Compiled code

Besides these disk files, there is one data base file that has all of the pertinent information about each program. This information includes library name, program name, author's initials, version date, key words, source code data, capsule data, and specification data.

B. Procedures

The software library facility uses the library files to control program development. A program under its control has two names. The first name is the run name. This is the name that is associated with the program for application identification. The second name is more or less an internal name that associates a "logical structure" to the program name. This name is made up of 6 characters. In this manner, programs can be categorized such as "system/facility/function" where the digit pairs represent categories.

It is suggested that a physical file folder be kept for each program in the software library. The enclosures should be:

current capsule document
current program specification
current program source listing
current program cross-reference listing
program change listings and submittal forms

The library facility is designed to restrict the access to the software library. Programmers can withdraw programs for development but they can not resubmit the programs directly. Each program is "password" protected. The librarian has the responsibiltiy for keeping the software library in working order. This person must ensure that programs entering the library contain all required documentation.

III. Program assignment

The project director first defines the requirements of a program or a group of programs. A "program specification" document is created. The library facility is used to assign programs to a programmer or programmers. The specification document is saved and a skeleton program is created in the programmers account. This skeleton program can contain any standard code that is desired. The library facility will use lines 1 and 2 of the source code.

During the assignment of programs, the following information is acquired:

development account
developer's initials
library name
program run name
key/words and phrases
heading names for "system/facility/function" to which this program belongs

Once this information has been obtained, the program becomes part of the software library.

IV. Program creation and submission

After the program has been assigned and created, it can be submitted to the library. In order to submit a program to the library, there must be a capsule document accompanying the program. The capsule document explains the purpose and usage of the program. The program author is responsible for the creation of the capsule. When the program is changed in any way, the individual changing the program must also update the capsule.

Each capsule document has five sections:

A. A heading line containing the program name.

B. An overview which is a short description of the program.

C. A user's guide that explains the operations of the program and includes a brief and useful demonstration of the program. This section is read by the non-technical person who is interested in learning how to use the program.

D. A reference guide which contains "in-depth" explanations and examples of the program. This section is directed at systems level persons.

E. Technical notes that contain file descriptions, program notes, file formats, warnings, and other such aids to programmers.

The programmer presents the program and capsule to the

librarian for submissiom. The librarian reviews the program to insure that it conforms to the programming standards of the organization. If it does, the program and capsule can be submitted to the library.

During the submittal process the library facility automtically generates a cross-reference listing of the program. After the program has been submitted, the library in conjunction with the physical file folder should contain:

program specification
program source listing
program cross-reference listing
submittal form
capsule document

After the first submission of a program, all future development is considered to be "program maintenance".

V. Program maintenance

The software facility provides an automatic method of reporting program bugs. When a bug is found in a program contained in the software library, the librarian under program control enters the report into the library. The bugs can then be listed and the maintenance be assigned to various programmers. At any time the program director has access to the bug assignment information such as:

date entered
description
initials of individual reporting the bug
initials of individual assigned to the bug
debugging account

When a program needs maintenance, the programmer withdraws it and/or related documents from the software library. While a program is out for development, the old version remains in the library and can be used for backup if need be. No other programmer is able to withdraw this program from the library.

After the program has been modified, the programmer presents a resubmittal form to the librarian along with the any updated documentation. One important piece of information contained on the resubmittal form is the "reason for change". This is an aid in following the updates of particular programs.

If all the information is in order, the librarian resubmits the program and/or documenation to the software library.

During the resubmittal process the library facility creates a cross-reference listing of the new program and compares it to the cross-reference listing of the old program. A program change listing is created. This listing states the lines that where changed.

The librarian updates the physical program file as follows:

program listing (replace - destroy old one)
cross-reference listing (replace - destroy old one)
program changes (add to old one(s))
resubmittal form (add to old one(s))
capsule document (replace old version)
program specification (replace old version)

VI. Manuals

By using the library facility manuals of all types can be generated and maintained. Manuals are the combination of two different sets of documents. The first set of documents are the program capsules that have been created by the programmer. The second set of documents are text files which contain descriptive text information and references to existing capsule documents. The software facility combines portions of the capsule documents (as requested by the text documents) with the text documents to produce a manual.

Capsules -- As previously mentioned a capsule is divided up into various sections. One way to section a capsule document is:

program name
overview
user's guide
reference information
technical notes

Other sections can be added if desired. The only requirement is that a section be identified.

Text files -- These files are created for whatever manual is to be generated. A good method to use is to have a text file for each chapter of a manual. In such files the descriptive text information is intermingled with references to capsule documents. These references can identify all of the capsule document or just portions or the capsule. In this way, text files can be written that contain for example user information or system overview data.

Generate the manual -- the library facility will then be used to find a text file. The references to the capsules in the text file will be resolved by removing the reference and inserting the capsule information of the desired section(s). This process results in the creation of a document suitable to be used as a manual.

Advantages --

The final manual will have information taken directly from the capsule. Each time the new version of the manual is generated, the data in the manual will represent the most recent changes to the capsules.

When changes are made to programs and capsules, it is not difficult to produce a new manual. By using the library facility, a new manual can be created with a minimum of effort.

Different types of manuals can be created by using the same program capsules.

VII. Concluding remarks

A software library facility will aid in the development of application programs and maintenance of a documented system by helping the user to avoid many potential problem areas.

Program specification and responsiblity is recorded.

The programs must conform to programming standards to be entered into the library.

Reports can be generated at any time to show the status of programs.

Programs in the library are compared to existing programs in designated accounts. The programs not in the library are reported.

"Bug" management and program maintenance is part of the library facility.

Programs can be revised by only one programmer at a time.

Parts of a manual include program capsules. These capsules are checked to insure that the programs do what the capsules state, and thus the chances of program-manual mismatches are reduced.

Manuals are generated by the library facility. If a capsule is changed, a new manual can be generated without a major editing cycle.

SYSTEM ACCESS MANAGEMENT

Presentor: Errol E. Ethier
Wachusett Regional School District
Holden, Massachusetts 01520

Day: Monday, Nov 28
Time: 4:15 - 4:45 p.m.
Room: Golden West

Since its inception the RSTS operating system has continually developed into a sophisticated, flexible and widely-used piece of software. Recent enhancements to RSTS have included the creation of four sub-systems: batch, spooling, backup, and error logging. This paper will present arguments for the development of another sub-system which monitors and controls access to the system; the rationale for the creation of the sub-system, the design considerations to be incorporated, and the the structure of one possible implementation will all be discussed.

The initial design of the RSTS operating system centered around the concept of making system resources available to all users on a shared basis; continued improvements to the software have reinforced that concept by making the sharing of system resources more flexible and efficient. The original designers of RSTS while recognizing the advantages of resource sharing also recognized the concurrent need to provide system users with both system access and file structure mechanisms which would guarantee the individual user private access to both the system and to the data stored on the system. This mechanism was implemented mainly through the use of individual accounts with accompanying passwords and file protection codes.

The growing sophistication of RSTS has seen its wide adoption into many diverse computing environments, many of which are involved with confidential and proprietary data as well as large scale remote terminal and system access; such installations usually require controlled access to the system and want that access to be both flexible and secure. While the development of RSTS has progressed significantly in many areas, there has been no significant improvement in the system or file access mechanisms; development of these areas of the software has not kept pace with the increased sophistication of the remainder of the software.

The RSTS software requires improved designs both in the system and file access mechanisms comparable to improvements made in other areas of the software. These designs must be incorporated as an integrated part of the software, not a patchwork to existing software. These designs should be based on system access principles which will provide a flexible, effective and secure implementation.

The paper to be presented will address the specific issue of system access, and will list and discuss major design principles that should be included in any implementation to allow effective monitoring and controlling of system access. The system access model to be described controls user access to the computer via both terminals and batch operation and significantly increases the security of the system by allowing many degrees and levels of login access to be defined by a system manager. Access to the system can be controlled by many factors including day, time, terminal, accumlulated usage, etc.

Features new to RSTS (but found in other non-DEC operating systems) are included in the design specifications: login preemption, password encryption, one-time passwords, use of optional keywords (as opposed to account numbers), chaining to user-specified programs from login, logging of all access operations (logging in, logging out, killing of jobs, system shutdowns and startups), threat monitoring, etc. The unique aspect of the model is that user access is controlled not only during the login procedure but is monitored and controlled throughout the user's continued access to the system and eventual logout. Implementation of this system access model requires that certain features of the existing system access mechanism be discarded or changed.

In order to test the feasibility and effectiveness of a system access mechanism under RSTS and to allow its refinement, an implementation of a system access sub-system has been designed and written in BASIC-PLUS. The implementation consists of 6 components: 5 programs and 1 data file. The data file consists of several types of records which define general system access parameters, individual account and terminal parameters, and password lists; data in this file is in binary format and encoded.

The programs that comprise this system access sub-system and their general functions are identified below. NOTE: The LOGIN and LOGOUT programs while retaining their traditional names are not the DEC-supplied programs.

LOGIN This program performs the initial authorization, identification, authentication and accounting decision-making functions of the system access mechanism. (This program also performs certain utility functions: setting of the user run-time and privilege level characteristics, printing of system and project notices, printing of user's mail if requested, and chaining to a user program.; this program also controls the access and running of some programs by a user not logged in.) This is the first component of the system access mechanism which the user encounters in attempting access (logging in).

LOGOUT This program performs the orderly, voluntary (in most instances) removal of the user from the system. (This program also performs certain utility functions: deletion of temporary and outdated files, printing of some system notices, and printing of user's mail if requested.) This is the last component of the system access mechanism which the user encounters in ending access (logging out) to the system.

LOGMON This program performs three main tasks: 1) monitoring each job on the system and enforcing removal of users from the system if a job is in an improper state or a job's connect time, run time or preemption exceeds a predetermined limit or expires, 2) removing or preventing given terminals and/or accounts from accessing the system during all or portions of a given operating session, and 3) logging of all system access events.

LOGCTL This program allows the system manager to control the system access mechanism by sending operational commands to LOGMON and/or altering the system access data file.

LOGFIL This program creates, updates and restructures the system access data file and allows the system manager to enter, delete, close and suspend user accounts, set the access and run-time characteristics and privileges of each account, specify system access parameters, and obtain reports on the characteristics and status of user accounts. At the system manager's option, a user may use this program to change limited characteristics of the user's own account.

This paper will discuss the model on which the implementation has been based, the features and operation of the implementation, and an analysis of the level of security provided.

Day: Tuesday, Nov. 29
Time: 8:30 - 9:00
Room: Cabinet

RPL11 - A Report generator for RSTS

T.J. Barnard, Independent.
Developed at Airfix Industries Ltd, U.K.

This paper describes a report generator for RSTS which generates small (typically 4K) programs which run 3-10 times faster than BASIC+. It has a powerful high-level language for declaring report formats, and a number of built-in file handlers to deal with most RSTS type file structures. In addition it offers the choice of two indexed access methods, one of which handles chained variable length records, the other allowing indexed sequential access with multiple keys and partial key extraction. When both these access methods are combined,RPL can be used as a powerful database management system.

Decision tables are used to specify record selection and other procedural logic where needed. These are compiled into directly executable PDP11 instructions by the RPL compiler, and run under the control of the RPL Run-Time System. Equal key compaction for totals only type reports is carried out by the built in sort in RPL, which runs 40 times faster than DEC's XQWIK. We hope yet to better the OMSI sort time of 50 times faster than XQWIK!

Pointers are a feature of the RPL language which allow for variable length record handling and character manipulation. There are equivalent operations in RPL to all the main BASIC+ string functions such as INSTR, CVT$$, XLATE etc. ,and improved versions of verbs like INPUT, GET, and PUT.

COMMERCIAL SUB-SIG

CHAIRMAN: Stuart M. Katz

DAY: Tuesday November 29th, 1977

TIME: 1:30 to 2:30

PLACE: Golden West Room

The purpose of the Commercial SUB-SIG session is to provide a time for the commercial users of RSTS/E to meet and discuss problems, ideas, and solutions.

This session will provide the now famous "Swap Table" upon which software and/or servcies which you wish to make available to other users may be displayed. (Be reminded that if you are bringing material, our numbers continue to grow.) The swap table will be open for set up prior to the session and the last 5 minutes of the session will be provided for users to examine the tables' contents.

The speaker for this session will be John Morand, treasure and general counsel for COMTECH. He will speak on contract law and how it effects:

Hardware
Software
Services

At each of these sessions in the past we have tried to have open discussion on how certain problems have been solved. Such topics have included remote printers and the difference between RSTS and COS 500. The bottom portion of this form is provided to help in determining which topics are of interest to the most people. If you have a preference please write it down and place it at the speakers table at the start of the session.

See you all there.

TOPICS FOR DISCUSSION

1) _____

Do you have a solution Yes/No

2) _____

Do you have a solution Yes/No

3) _____

Do you have a solution Yes/No

Name _____

ESTABLISHING AN INTERNAL TIMESHARING SERVICE

Presentor: Kenneth Ross

Day: Tuesday, November 29, 1977
Time: 2:30 - 3:00 p.m.
Room: Golden West Room

The last few years have seen the mini-computer grow to the point where it has become feasible to use as a general purpose timesharing machine within the environment of the large company. The replacement of services purchased from the large commercial timesharing vendors can save a company large amounts of money and can itself justify the purchase of such a machine.

In reality, the commercial timesharing services have provided much more than an interactive computer along with its associated languages. Specialized applications, languages, networks and a great deal of user support are all factors that have contributed to their success. The company attempting to provide general timesharing within their own firms must consider these and other factors if they hope to be successful.

This paper attempts to discuss the significant factors that can make in-house timesharing a success as well as some practical experiences gained in working with some of the companies.

EDUCATIONAL RSTS SUB-SIG

I. Purpose of the SIG

A. Provide a "get acquainted" session for educational RSTS users.
B. Coordinate input to RSTS SIG from educational users.
C. Focal point for software exchange.
D. Focal point for software and hardware request.
E. Provide for discussion of RSTS concerns from educators.

AGENDA
Fall 77 DECUS Symposium
San Diego, California

DATE: Tuesday, November 29th
PLACE: Senate/Committee Rooms
TIME: 3:15 - 4:15 PM

I. Introduction of persons in attendance.
General Format:
A. Name
B. Organization
C. Immediate interest
D. Resources to share

II. Report from CAI users.
A. DECAL-Charles Mustain
B. DECAL-George Wyncott
C. AIB-David Hamlin
D. Others

III. Report on statistical packages.
A. COSAP-II-Michael Hall
B. SPSS-Graham Kimble
C. Others

IV. Report on software systems operational in an educational environment.
A. WISE-Mark Grundler
B. Others

V. Educational RSTS concerns-Wish List.
Charles Mustain coordinator

VI. Planning for the spring meeting.

VII. Highlight sessions of interest-Security-DeAnza/Mesa Room
Wednesday, November 30, 8:30 - 9:30
Buzz session

Concepts for Systems Operation

or

How to Help the Dumb End User Survive

Presenter: Stuart M. Katz

Day: Tuesday November 29th, 1977

Time: 3:15 to 3:45

Place: Golden West Room

The purpose of this workshop is to explore various concepts whose purpose are to help create a user independent RSTS/E shop. The workshop will begin with some current prcedures being used and then the meeting will be opened for participation by the users present. Quite often, those very simple concepts which are used everyday are assumed to be too obvious to suggest to others. Just the opposite is true! Come and share your ideas and maybe you will take home a few good concepts.

CCS installs customized RSTS/E systems for small and large shops. A small shop is defined as an installation of a PDP-11/40 or PDP-11/34 with 40 megs or less of disk storage. This type of shop normally has no professional DPer and hopes to have none in the future. A large shop therefore is defined as a larger CPU with more disk and some kind of professional staff. For those small shops the need for user independence is obvious. For the large shops user independence is usually the desired result.

User independence can be defined as the capability for any user to run his or her application without being anymore familiar with the system then how to sign on. (We have never utilized any special security.) Once signed on, the system should lead the user through each step. The following are some of the things we have done to help the users and these are the topics I will describe briefly.

1. Index to Index system flow

2. Conversational mode input and process control

3. Job streaming for month end reports utilize on-the-fly batch control file generation

4. Standardization in system operation

If you have reservations when I say that the small shop can be run with no DPers, well, I really mean it. You obviously need someone that can load a tape, remove a disk, maybe even que up a spooled report, let alone start and stop the system. This person in the small shop has been a secretary, an ex-key punch operator, etc. In order to help this person, usually labeled the system manager, we have tried to use the following concepts.

1. Batched backup (both 5C & 6B) using control files generated on the fly.

2. Queing spooled reports. (Both from disk & magtape)

3. In shops where remote sites are available we have used a file called an account definition file for keeping track of what device is the remote printer, what disk that account's files are on etc.

4. Conversational input required a great deal of verification. We have utilized a very simple method to store this data. This is defined as the standard array.

These are the basic concepts that I will present and then open the floor for discussion. I think this session can be most useful and with your participation you can help further a good exchange of ideas.

MAJOR INGREDIENTS OF A GENERAL ACCOUNTING

PACKAGE UNDER RSTS

Presentor: Eric E. Beaverstock
Intelligent Systems Corp.
460 Totten Pond Road
Waltham, Mass. 02154

day: Tuesday Nov 29th, 1977
time: 3:45 PM - 4:15 PM
room: Golden West

Time sharing under RSTS provides a fertile environment for the implementation of a general accounting package. This presentation describes implementation techniques, key elements, and critical issues involved in the development of such a package. The four major accounting areas:

General Ledger

Accounts payable

Accounts receivable

Payroll

are singled out for discussion which includes file structures and problem areas. The implementation of such an accounting system needs commonly shared utilities. The features of such utilites are also described.

The ease with which the user is able to initialize and operate the system is an important element that is included in the discussion.

Information presented has been acquired through the actual creation of a general accounting package by Intelligent Systems Corporation. The data presented results from experiences acquired through the implementation of the package at various user sites.

I. Overview

The major areas associated with a general accounting package are:

General ledger

Accounts payable

Accounts receivable

Payroll

Each area can be implemented with its own set of programs and thus make up unique facilites.

A. The general ledger facility includes capabilities that allow the user to:

Initialize and maintain the general ledger accounts

Post transactions in an interactive mode or automatically post transactions for the user.

Error check all ledger numbers

Review individual accounts

Customize financial reports

B. The accounts payable facility includes capabilites that allow the user to:

Initialize and maintain the vendor file

Post transactions to the general ledger and vendor file (and journalize).

Error check posting data

Review individual vendor accounts

Obtain cash requirement reports

Selectively disburse cash

Write checks

C. The accounts receivable facility includes capablilites that allow the user to:

Initialize and maintain the customer file

Post transactions to the general ledger and customer file (and journalize).

Error check posting data

Review individual customer reports

Obtain an aged trial balance

Generate invoices and monthly statements

D. The payroll facility includes capabilities that allow the user to:

Initialize and maintain the personnel file

Post transactions to the general ledger and personnel file (and journalize)

Error check posting data

Review individual employee information

Produce various reports such as W-2 or 941

Selectively disburse payroll

Write checks

These major facilities can be used alone, but in most cases, the general ledger facility will be used along with any combination of the other facilites.

II. Shared utilites

A. Data flow

Although a general accounting package does have separate application areas, there is a common flow of data through the system. Information is first captured at the keyboard by means of data entry programs. Once all of the data has been entered and validated by the posting clerk, updating takes place. Journals describing the posting information are required by all facilites.

B. Support activites

Posting -- nearly every company, department, or institution has certain bookkeeping or posting procedures which are fairly unique. Rather than continually changing programs to handle these varying needs posting routines can be implemented with the aid of posting attribute files. One basic post program is able to read an attribute file to acquire necessary posting data for a particular application. The type of information contained in these posting attribute files include:

Input requirements
Input data error checks
Output file information

The technique allows a posting program to be driven by attribute files. The files can be changed easily.

File updating -- The master queue capability consists of a set of programs that control the updating of the data files. For user activites that are to modify the master data files, posting records are created. These posting records are placed into a queue. The records are posted to the master data files by automatic programs which add, delete, and change records in the various data files in a background mode so that corruption due to human error or computer malfunction is kept under strick control. When the automatic posting programs succeed in posting changes to the data files, the posting records are removed from the queue and transferred to a backup media. If a problem arises, another program will re-run the update information against user backup files of the previous day so that the data will not be lost and there is no need to re-enter data.

Journalizing -- All of the posting programs create text files containing posting data. A generalized utility program allows the user to select the journal desired from a menu. This program will do the necessary housekeeping tasks and run the correct format programs to insure that the proper formatted journal is created. In this way, the user only has to interface with one program rather than a separate program for each journal.

III. General ledger

A. The main files in the general ledger facility consist of the a summary file and a set of transaction files.

There is one summary file for each fiscal year. Each account number in the chart of accounts has a record in this file. The type of data deals with chart of account summary information such as monthly balances, or starting balance.

There is one transaction file for each month of the fiscal year. This file is used to store the tranactions which are posted to the general ledger.

B. Customized financial reports

Since reports vary from application to application, it is advisable that the user be able to design and implement procedures that allow the user to produce customized financial reports. A report generating program is driven by a file that is created by the user. This file contains commands that dictate to the report generating program where the data is coming from, and where the data is to go. The balances for the general ledger accounts can be retrieved easily and summarized in many different ways. Balances may be obtained for any time period desired.

IV. Accounts payable

A. The main file in the accounts payable facility is the vendor file.

The first 11 or so fields make up the header portion of of a vendor record. The remaining fields may repeat many times and are used to store "line item" data.

Header - the data stored in this section includes vendor name, address, phone, and attention name.

line items - these fields record puchase order date and number, invoice date and number, credit information, and payable data.

B. One of the major problem areas dealing with accounts payable is the check formats. One approach is to have a fixed format and require the application to conform to the format. A much better approach is to allow the user to design the check layout(s). A file can be constructed which contains commands that will drive a general check printing program. The data in the file includes line location and tab position along with data retrieval instructions. The check printing program only has to know the name of the file that is to have the instructions of the particular check writing session.

V. Accounts receivable

A. The main file in the accounts receivable facility is the customer file.

The first 11 or so fields make up the header portion of a customer record. The remaining fields may repeat many times and are used to store "line item" data.

Header - the data stored in this section includes customer name, address, phone, and attention name. It may be desirable to include provisions for billing information which might differ from the regular customer address.

line items - these fields record sales data such as date, description, invoice number, and amount.

B. One area of attention not only for this facility but for all facilites which have files that capture repeating groups of line item data is the problem of how much data should be retained in the file. In all cases, the user must be able to run a program which deletes line items from a file. This application has the added problem that line items are to be removed only if the line items are no longer open (there is a zero amount owed on the particular invoice). Another requirement is that no items are to be purged unless they were posted at a date earlier than the date entered as an user input parameter. In this way, a certain amount of customer history is kept on the computer at all times.

VI. Payroll

A. The main file in the payroll facility is the personnel file.

The first 20 or so fields make up the header portion of an employee record. The remaining fields repeat many times and are used to store data about the pay periods.

Header -- the data store in this section includes employee name, address, phone, social security number, pay data, tax data, and deduction information.

pay period -- the dates on which payroll information is entered into the system.

pay data -- infomation such as gross pay, net pay, deductions, taxes, and data on hours worked, earned amounts, and project numbers.

B. One of the problems with the computation of an employee net pay has to do with the taxes. The Federal tax withholdings are well defined, but the state withholdings differ from state to state. Other deductions for an employee vary with the employee. One way to ease these types of problems is to include in the header portion of the employee record tax and deduction data. This data includes a description of the deduction and a method for computing the amount. The main methods of computation are:

table look-up (tax tables)

function (FICA)

constant amount

The user will still have to define a table look-up routine if that is what is needed, but the hooks will already be in the program and the implementation should not be too difficult.

VII. User interface

A. One of the requriements of an accounting package is that the user have confidence that there is system protection against unwanted or accidental program usage by unauthorized personnel. Each program in such a package should require that the user enter a valid "program-password" before running a program. If an incorrect password is entered, the program aborts to a "NONAME" status. This will to protect the installation from unauthorized or inadvertent usage. The program-passwords can be stored in an array which only the system manager can modify. The modification process must be easy to perform, and a single program should be able to have multiple valid program-passwords to allow greater fexibility in determining which users may run which programs.

B. Another requirement of an accounting package is that the user

manuals must be easy to follow, and accurate. Nothing is more frustrating than to find that programs do not conform to the descriptions in the manual. The problem can be reduced by having organized control over the program development. This can be accomplished with a software library scheme which is used to organize, maintain, and control the software library. One feature of such a system is the use of a program that captures "sample dialogues" between the computer and the user. The program records the dialogue as it occurs, and this information can be inserted directly into the manual. The program also underlines all characters typed by the user to distinguish them from the computer output and also inserts a "<CR>" to show a single carriage-return typed by the user. This method of reproduction provides a reliable guide for the user.

VIII. Concluding remarks

A. For the implementation of any software system it is important to identify activites which are shared by all facilities. Once these activites have been singled out, programs can be written that are general purpose in nature and task supportive.

B. When an application calls for report generation, it is advisable that the system design allow for the user to participate in the output report definitions. This can be accomplished by means of file driven programs.

C. Check writing like report generation requires the user to participate in the output definitions. Although the application is a little different, the same type of approach using file driven programs can be used.

D. During the design of a system, it might not be possible to complete the implementation of all user applications. The important design feature is that there are easy ways by which code may be tailorized to fit a user's particular need without a major re-design effort.

E. Once the system has been implemented to the satisfaction of the designer(s) the user manuals and documentation must be presented in a manner which will be satisfactory to the user. This can be accomplished if the programs conform to the descriptions in the manuals.

A THING CALLED ZAP

Presentor: John A. Christensen
Idaho Falls School District

Day: Tuesday, Nov. 29
Time: 4:15-4:45
Room: Senate/Committee

Idaho Falls School District has been using a RSTS system for 5 years as an instructional and administrative tool. Both applications are running simultaneously. This requires some special system management because of the confidentuality of administrative jobs and the inexperienced and sometimes malicious student user.

We have developed a system of management centered around a program called ZAP. ZAP has many functions such as systats and quota, run burst, priority and swap max changing, but most important it allows for 'zapping' users. If a user is caught doing or using something he knows he shouldn't, the program can be run to 'zap' him. When the program 'zaps' a user it first detaches him, then changes his password and kills his job. If a user runs a program with a protection code of 233 he is automatically 'zapped' if he is not logged into an appropriate account. A file keeps track of who's 'zapped' and the user may be 'unzapped' after he has suffered through a detention period.

Along with ZAP there are many other programs which play an important role in our system management. There is a complete accounting package controlled by a job called WAKE. WAKE does many things also like protect psuedo keyboard.

We have a complete password changing system along with a method for assigning account numbers based on the type of user. 'Zapped' users may not change their passwords. Some of DEC's cusps have been modified to aid in system management. An operator-user communications program TALK is also part of our system. It has unique features.

All of these utilities and others that aid our system management will be discussed in this session. Non-educational users may find our system and utilities helpful. They are our contribution to EDUSIG and the RSTS SIG.

MAPS - AN INTERACTIVE FINANCIAL MODELING LANGUAGE

Presentor: Kenneth Ross

Day: Tuesday, November 29, 1977
Time: 4:15 - 4:45 p.m.
Room: Golden West Room

This paper discusses the concept, development and marketing of MAPS, an interactive financial reporting and modeling language that runs on the PDP-11 using RSTS/E. MAPS is a "problem" oriented language that deals with the manipulation and reporting of financial matrices. It is oriented toward the non-technical user, but people with a programming knowledge can use MAPS to develop comprehensive financial systems. MAPS is written entirely in BASIC-PLUS (with a BASIC-PLUS 2 version).

Day: Tuesday, Nov. 29
Time: 4:45 - 5:15
Room: Golden West

FORTRAN under RSTS

This session will be an interactive get-together of people using, or interested in the use, of FORTRAN under RSTS. We will discuss common problems and offer some hints and kinks. Some of the topics we will talk about are: New FORTRAN systems for RSTS - V2 and IVPLUS, RT11 SYSLIB routines, RT11 FORTRAN extensions, monitor calls from MACRO, poor FORTRAN SPR service, and others.

SYSTEM SECURITY IN AN EDUCATIONAL ENVIRONMENT

(A Tutorial)

Presentor: Errol E. Ethier
Wachusett Regional School District
Holden, Massachusetts 01520

Day: Wednesday, Nov 30
Time: 8:30 - 9:30 a.m.
Room: DeAnza/Mesa

This session will consist of a tutorial dealing with system security followed by a question and answer period. While the emphasis will be on system security within the context of an educational environment, the concepts presented will be applicable to any environment. The tutorial will not dwell on an analysis of the weaknesses of any given operating system or model of hardware, but rather will analyze those components that make any system secure. The tutorial will overview both system access security and file access security, with the former topic receiving most attention.

Recognizing that most educational users cannot easily change hardware or software but must live with the hardware and software available to them, the tutorial will suggest administrative practices and simple software modifications which when adopted greatly increase security on any system. Although some useful stop-gap measures will be presented, the tutorial will emphasize that an integrated, total-system-design approach to security is ultimately the only reliable approach.

The tutorial will deal with the following topics:

1) Security - what it is and is not

2) The necessity of a secure system

3) Evaluating potential threats to a system

4) Components of a secure system

5) Determining the existing level of system security

6) Planning for the required level of system security

7) Implementing a secure system

8) Specific administrative practices for increasing and enforcing system security

9) Useful software techniques for increasing system security

PRESENTER: Robert A. Branton, Jr.

DAY: Wed., Nov. 30
TIME: 2:30 - 3:30
ROOM: Golden West

ON-LINE A/R ENTRY UNDER RSTS

This paper will cover the design of the on-line accounts receivable entry programs I have developed for Omega Optical Company. The original entry system which was written two years ago has been revised to include:

1. The features of RSTS V6B, in particular Echo Control.

2. A detached index lookup program that handles all the searching for customers so that verificatia can be made (INDEXC).

3. A detached program that records batches of invoices when they are released and additionally makes a continual backup of all file blocks updated by the A/R package (BALOOK)

I will cover the improvement in system performance that these changes have made.

I N D E X C

THE "INDEXC" PROGRAM IS RESPONSIBLE FOR THE VERIFICATION OF CUSTOMER NUMBERS FOR THE A/R ENTRY PROGRAMS, "TENEW" AND "CENEW", AND FOR FINDING THE LOCATION OF CUSTOMERS IN THE CUSTOMER FILE, "ARPACK:ACOUNR.OME". INDEXC RUNS DETACHED FROM ANY TERMINAL AND COMMUNICATES WITH THE ENTRY PROGRAMS VIA SEND/RECEIVE MESSAGES. IT IS AUTOMATICALLY STARTED WHEN THE SYSTEM IS STARTED AND IS GIVEN A VERY HIGH PRIORITY (+16) SO THAT ALL REQUESTS FOR CUSTOMERS ARE HANDLED AS QUICKLY AS POSSIBLE.

THE SEND/RECEIVE COMMUNICATION BETWEEN THE ENTRY PROGRAMS AND INDEXC IS ACTIVATED WHENEVER A CUSTOMER NUMBER IS ENTERED FOR A TRANSACTION ON AN ENTRY TERMINAL. THE CUSTOMER NUMBER IS SENT TO INDEXC FOR VERIFICATION WHILE THE REST OF THE TRANSACTION IS BEING ENTERED ON THE ENTRY TERMINAL. INDEXC TAKES THE RECEIVED CUSTOMER NUMBER AND SEARCHES THE CUSTOMER INDEX FILE, "ARPACK:ACCIND.OME", FOR A MATCH ON THE CUSTOMER. IF THE CUSTOMER IS FOUND, HIS LOCATION IN THE CUSTOMER FILE IS SENT BACK TO THE ENTRY PROGRAM. IF IT IS NOT FOUND A ZERO IS SEND BACK TO THE ENTRY PROGRAM. BY USING THIS TECHNIQUE THE CUSTOMER'S LOCATION IS USUALLY AVAILABLE FOR THE ENTRY PROGRAM BY THE TIME THE REST OF THE TRANSACTION HAS BEEN ENTERED.

INDEXC ALSO RECEIVES MESSAGES FROM "ACOUNR", THE CUSTOMER FILE MAINTENANCE PROGRAM. WHENEVER, THE CUSTOMER FILE INDEX IS CHANGED BY ADDING NEW CUSTOMERS, ACOUNR SENDS INDEXC A MESSAGE TELLING IT TO REBUILD ITS IN-CORE INDEX INFORMATION.

THE FORMATS OF THE MESSAGES ARE AS FOLLOWS:

A. REQUESTS FROM : TENEW AND CENEW

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(0)+'INDEXC'+
          STRING$(10,0)+CHR$(COMPANY%)+CUSTNUM$)
```

WHERE:

COMPANY%	- OPERATION CODE OF CUSTOMER (1,2,3,4,91,92,93, OR 94)
CUSTNUM$	- CUSTOMER NUMBER TO LOOKUP (6 DIGIT NUMBER)

B. ANSWERS TO: TENEW AND CENEW

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(JOB%)+
          STRING$(16,0)+CVT%$(C5%)+CHR$(COMPANY%)+CUSTNUM$)
```

WHERE:

JOB%	- SYSTEM JOB # OF THE ENTRY PROGRAM
C5%	- CUSTOMER'S LOCATION IN THE CUSTOMER FILE OR 0 IF THE CUSTOMER WASN'T FOUND BY INDEXC
COMPANY%	- OPERATION CODE OF CUSTOMER (1,2,3,4,91,92,93, OR 94)
CUSTNUM$	- CUSTOMER NUMBER OF THE REQUESTED CUSTOMER (6 DIGIT NUMBER)

C. MESSAGES FROM ACOUNR (NO REPLIES ARE SENT BACK)

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(0)+'INDEXC'+
          STRING$(10,0)+CHR$(MSGCODE%)+CUSTNUM$+CVT%$(BLOCK%))
```

WHERE:

MSGCODE%	- 100 (INDICATES A MESSAGE FROM ACOUNR)
CUSTNUM$	- THE HIGHEST CUSTOMER# IN CHANGED INDEX BLOCK
BLOCK%	- THE INDEX BLOCK THAT CHANGED

B A L O O K

THE "BALOOK" PROGRAM IS RESPONSIBLE FOR THE CONTROL OF THE TRANSFER OF A/R BATCHES TO THE DAILY A/R FILES FROM THE TEMPORARY FILE INTO WHICH THEY ARE ENTERED. IT ALSO OVERSEES THE BATCH NUMBERS ENTERED TO INSURE THAT A BATCH CANNOT ACCIDENTALLY BE ENTERED TWICE. THE PROGRAM IS STARTED UP AUTOMATICALLY AT STARTUP TIME AND IT RUNS DETACHED FROM ANY TERMINALS.

BALOOK COMMUNICATES WITH THE ENTRY PROGRAMS, "TENEW" AND "CENEW", THROUGH SEND/RECEIVE MESSAGES AND VIA THE FIRST BLOCK THE TEMPORARY ENTRY FILE, "ARPACK:BATCH.DUM". WHEN AN OPERATOR BEGINS PROCESSING A BATCH AND ENTERS A BATCH NUMBER, A MESSAGE CONTAINING THE BATCH#, OPERATION, AND DATE IS SENT TO BALOOK. IT CHECKS TO MAKE SURE THE BATCH HASN'T ALREADY BEEN ENTERED OR ISN'T IN THE PROCESS OF BEING ENTERED ON ANOTHER TERMINAL. AFTER THE CHECK IS MADE A MESSAGE IS SENT BACK TO THE ENTRY PROGRAM TELLING IT WHETHER THE BATCH CAN BE ENTERED OR NOT.

WHEN BALOOK IS NOT BUSY ANSWERING MESSAGES, IT CHECKS THE FIRST BLOCK OF THE BATCH.DUM FILE FOR COMPLETED BATCHES. EACH ENTRY PROGRAM HAS A 25 BLOCK SLOT IN THE FILE WHICH HOLDS UP TO 192 TRANSACTIONS AND IN THE FIRST BLOCK OF THE FILE ARE TWO CHARACTER FLAGS FOR EACH SLOT.THESE FLAGS ARE IN CVT$% FORM AND CORRESPOND TO THE KEYBOARD#'S. A KEYBOARD'S FLAG SPOT IS (KEYBOARD#-8), SO KEYBOARD# 10 IS ASSIGNED FLAG 2 FOR EXAMPLE. TO GET TO THIS FLAG THE STATEMENT: FIELD #9,2% AS D$,2% AS FLAG$ WOULD BE USED. THE FLAGS CAN HAVE ONE OF THREE VALUES: 0- THE KEYBOARD IS NOT BEING USED FOR ENTRY, KEYBOARD#*2- A BATCH IS BEING ENTERED,1- A BATCH HAS BEEN COMPLETED AND NEEDS TO BE PUT INTO THE DAILY FILES. WHEN BALOOK SEES THE COMPLETED CONDITION IT TAKES THE INFORMATION IN THE KEYBOARD'S SLOT IN BATCH.DUM AND MOVES IT INTO THE DAILY FILES AND THEN ZEROS THE KEYBOARD'S FLAG SO THAT ENTRY OF OTHER BATCHES ON THAT TERMINAL MAY CONTINUE.

IN ADDITION TO THE MESSAGES FROM THEN ENTRY PROGRAMS BALOOK RECEIVES TWO SPECIAL MESSAGES FROM "EBATCH". ONE MESSAGE TELLS IT THAT A BATCH HAS BEEN DELETED FROM THE DAILY FILE AND CAUSES BALOOK TO DELETE THE BATCH# FROM ITS LIST OF RECORDED BATCHES. THE OTHER MESSAGE TELLS BALOOK THAT THE DAILY FILE HAS BEEN ZEROED BY EBATCH AND THAT IT IS TO CLEAR ITS LIST OF RECORDED BATCHES SO A NEW DAY OF PROCESSING CAN BE STARTED.

THE FORMATS OF THE MESSAGES ARE AS FOLLOWS:

A. REQUESTS FROM : TENEW AND CENEW

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(0)+'BALOOK'+
          STRING$(10,0)+CHR$(TYPE%)+CVT%$(OPERATION%)+
          CVT%$(BATCHNUM%)+CVT%$(BDATE%)+CVT%$(0)+
          CVT%$(NINESPOT%))
```

WHERE:

```
TYPE%       - 1(INDICATING A MESSAGE FROM TENEW OR CENEW)
OPERATION%  - OPERATION CODE FOR THE BATCH
BATCHNUM%   - BATCH# TO CHECK
NINESPOT%   - KEYBOARD'S FLAG SUBSCRIPT( KEYBOARD#-8)
```

B. ANSWERS TO : TENEW AND CENEW

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(JOB%)+STRING$(16,0)+
          CHR$(1)+CVT%$(OPERATION%)+CVT%$(BATCHNUM%)+
          CVT%$(BDATE%)+CVT%$(P1%)+CVT%$(NINESPOT%))
```

WHERE:

```
JOB%        - JOB # OF THE ENTRY TERMINAL
OPERATION%  - OPERATION CODE FOR THE BATCH
BATCHNUM%   - BATCH# TO CHECK
BDATE%      - DATE OF THE BATCH (MMDD)
P1%         - ANSWER FLAG TO THE ENTRY PROGRAM
                    0 - BATCH IS "OK"
                   -1 - BATCH HAS ALREADY BEEN RECORDED
                   -2 - BATCH IS ALREADY BEING ENTERED
NINESPOT%   - KEYBOARD'S FLAG SUBSCRIPT( KEYBOARD#-8)
```

C. MESSAGES FROM : EBATCH (NO ANSWERS ARE SENT)

```
X9$ = SYS(CHR$(6)+CHR$(22)+CHR$(-1)+CHR$(0)+'BALOOK'+
          STRING$(10,0)+CHR$(TYPE%)+CVT%$(0)+CVT%$(0)+
          CVT%$(0)+CVT%$(0))
```

WHERE:

```
TYPE%       - TYPE OF MESSAGE FROM EBATCH
                   10 - DAILY FILES HAVE BEEN CLEARED
                   11 - A BATCH HAS BEEN DELETED, REBUILD
                        LIST OF BATCHES
```

TENEW SCREEN TWO FORMAT (BATCH INFORMATION)

```
B A T C H   I N F O R M A T I O N   ==>

                    OPERATION CODE(01)? __

                    ENTER BATCH # OR 0? ____

                    DATE (02.11)? _____

                    TRANSACTION TYPE (1)? _

                    OPERATOR (YOUR NAME) ----------

                    ENTER EXPECTED BALANCE ? ---------
```

YOUR RESPONSES ARE:

<CR> TO DEFAULT OR A NEW OPERATION CODE <CR>

BATCH # <CR> OR 0 <CR> TO END PROGRAM

<CR> TO DEFAULT OR MM.DD <CR>

<CR> TO DEFAULT OR NEW TRANSACTION CODE <CR>

<CR> TO DEFAULT OR NEW NAME <CR>

THE BATCH'S TOTAL <CR> -- WITH OR WITHOUT THE DECIMAL

OMEGA'S OPERATING PROCEDURES

TENEW SCREEN THREE FORMAT (ENTER TRANSACTIONS)

```
          BATCH#: 99                                        OPERATION#: 01
          ITEM#:  2                                         TRANS TYPE: 1
          L INV#: 123456                                    DATE:       2/11
                   TRANSACTION ==>
                   RECORD==>     ---
                           CUSTOMER#:          NAME:
                           INVOICE #:
                           AMOUNT    :
                           TRAY#     :
          PRESENT TOT:  $   10.00
          EXPECTED TOT: $ 1000.00
```

YOUR RESPONSES TO 'TRANSACTION==>' ARE:

0 <CR> = THE PROGRAM GOES TO THE RECORD THE BATCH SCREEN

S <CR> = ABORTS THE BATCH -- THE BATCH IS LOST & THE PROGRAM STOPS

BACK KEY = THE LAST RECORDED INVOICE IS DELETED

CCCCCC <CR> IIIIII.AAAA.CTTTT <CR>

WHERE:
CCCCCC = CUSTOMER NUMBER -- MAXIMUM SIX DIGITS
IIIIII = INVOICE NUMBER -- MAXIMUM SIX DIGITS
AAAA = INVOICE AMOUNT -- NO DECIMAL
CTTTT = ZERO OR TRAY CODE AND NUMBER -- SEE THE TRAY CODES
NOTE: THE PERIODS (DOTS) MUST BE PRESENT

YOUR RESPONSES TO 'RECORD==>' ARE:

<CR> = RECORD THE INVOICE

NO KEY = PROGRAM GOES TO THE CORRECTION ROUTINE

BACK KEY = REMOVES ONE ON THE SCREEN AND ALSO THE LAST RECORDED ONE

TENEW SCREEN FOUR FORMAT (CORRECTION)

IF THE RESPONSE TO 'CORRECT???' IS THE <CR> THEN THE FOLLOWING SCREEN IS DISPLAYED. IF THE RESPONSE TO 'CORRECT???' IS THE 'NO' KEY THEN THE INVOICE IS IGNORED.

```
          BATCH#: 99                                        OPERATION#: 1

          ITEM#:  2                                         TRANS TYPE: 1

          L INV#: 123456                                    DATE:       2/11

                0(STOP).1.CUST.2.INV.3.AMT.4.TRAY ---------------

                   CUSTOMER#:            NAME:

                   INVOICE #:

                   AMOUNT    :

                   TRAY#     :

          PRESENT TOT:  $   10.00

          EXPECTED TOT: $ 1000.00
```

YOUR RESPONSES TO THE CORRECTION OPTIONS ARE:

```
0 <CR>              = END THE CORRECTION - GOES TO THE RECORD OPTION

1.CCCCCC <CR>       = CHANGE THE CUSTOMER NUMBER - SIX DIGITS

2.IIIIII <CR>       = CHANGE THE INVOICE NUMBER  - SIX DIGITS

3.AAAA   <CR>       = CHANGE THE AMOUNT          - NO DECIMAL

4.CTTTT  <CR>       = CHANGE THE TRAY NUMBER     - ZERO OR CODE AND NUMBER
```

TENEW SCREEN FIVE FORMAT (RECORD THE BATCH)

```
          BATCH#: 99                                    OPERATION#: 01
          ITEM#:  2                                     TRANS TYPE: 1
          L INV#: 123456                                DATE:       2/11
          RECORD THE BATCH???     ___

                    FINAL TOTAL $ 1000.00
                    EXP   TOTAL $ 1000.00
```

YOUR RESPONSES TO 'RECORD THE BATCH???' ARE:

Y <CR> = RECORDS THE BATCH - PROGRAM GOES TO BATCH INFORMATION SCREEN

N <CR> = DOES NOT RECORD THE BATCH - PROGRAM ASKS 'CONTINUE BATCH?'

NO KEY = DOES NOT RECORD THE BATCH - PROGRAM ASKS 'CONTINUE BATCH?'

YOUR RESPONSES TO 'CONTINUE BATCH?' ARE:

Y <CR> = PROGRAM GOES TO THE ENTER THE TRANSACTION SCREEN

<CR> = PROGRAM GOES TO THE ENTER THE TRANSACTION SCREEN

N <CR> = PROGRAM GOES TO THE BATCH INFORMATION SCREEN

NO KEY = PROGRAM GOES TO THE BATCH INFORMATION SCREEN

DAY: Thursday, December 1
TIME: 10:15 a.m. - 10:45 a.m.
ROOM: DeAnza/Mesa

HOW TO USE RSTS/E: A USER-ORIENTED TRAINING PACKAGE

J. A. Hayes, Academic Coordinator
Computer Center
California State University, Northridge
Northridge, California

ABSTRACT

The California State University, Northridge Computer Center has four computer systems available to the campus community. One of the systems is the local, on-campus timesharing system, a 32-port PDP 11/45 running RSTS/E. The PDP 11/45 is strictly an instructional computer serving approximately 200 classes with more than 4,000 students.

In order to introduce novices to RSTS/E timesharing, a user-oriented training "package" was developed. This package includes two multimedia (slide/cassette) shows, on-line and hard copy user-oriented publications, microfiched RSTS/E manuals and a series of seminars.

INTRODUCTION

In any university environment, some sort of instructional support is necessary for serving the academic community in its use of computer systems. At California State University, Northridge the Computer Center has been strongly involved in this activity for more than ten years. Each year the Computer Center has provided more and better services to students and faculty using the various computer systems. Currently there are two local systems: a CDC 3300 running MASTER and a PDP 11/45 running RSTS/E. Two central systems, dual CDC 3300's running MASTER and a CDC CYBER 173 running NOS, can also be accessed by students and faculty.

A limited instructional support staff of six people plus student assistants serves the total computing academic community of some 9,000 students. With this rather large number of students to serve, it was necessary to develop training materials and various user-publications for the students and faculty to learn how to use the various computer systems. This paper, HOW TO USE RSTS/E: A USER-ORIENTED TRAINING PACKAGE discusses the various services offered to some 4,000 RSTS/E users.

COMPONENTS OF THE TRAINING PACKAGE

The components of the RSTS/E training "package" are designed to present materials to the user in a variety of media. Audio and visual methods are used separately and together. Multimedia (slide/cassette) and live seminar presentations are also used to provide a maximum learning environment. Publications and training media are aimed at the novice user (either student or faculty) as well as the more sophisticated user.

The components of the RSTS/E training package are:

- INTRODUCTION TO THE COMPUTER CENTER - A basic multimedia show to introduce the new user to Computer Center facilities and the various computer systems available.
- HOW TO USE RSTS/E TIMESHARING - A multimedia show specifically designed to introduce the user to RSTS/E timesharing.
- RSTS/E PUBLICATIONS - A variety of on-line and hard copy publications supplement vendor-provided manuals.
- MICROFICHED RSTS/E MANUALS - All RSTS/E manuals have been microfiched to provide easy access for students and faculty.
- SPECIAL TOPIC SEMINARS - Academic staff personnel will provide seminars on nearly any RSTS/E topic to both student and faculty users.
- SPECIAL INSTRUCTOR TUTORIALS - To better prepare faculty members to teach classes using RSTS/E, individual tutorial sessions are provided.
- STUDENT AND FACULTY CONSULTING - This is the follow-up support service offered to all users. Programming questions are answered, users are assisted in debugging, etc.

MULTIMEDIA PRESENTATIONS

Two multimedia presentations are usually shown to new users as a way of explaining the essentials of using one of the four computer systems available. After one or two classroom sessions in a computer using class, the instructor will arrange for the students to see two multimedia shows. All new users are advised to see the multimedia presentation, INTRODUCTION TO THE COMPUTER CENTER. This general slide/cassette show introduces users to the various Computer Center facilities, hours of operations, general services and the various computer systems available. The instructor then recommends a system specific multimedia presentation so students can begin their classroom assignments. One of these system specific multimedia shows is How To Use RSTS/E Timesharing.

How To Use RSTS/E Timesharing

The RSTS/E multimedia show is a synchronized slide/cassette presentation providing the new user with some background facts and sufficient information to either begin to write programs or to make use of library programs on RSTS/E. This multimedia presentation:

- places the RSTS/E timesharing system into perspective with the other computer systems available;
- briefly describes the hardware components;
- covers the concept of the operating system and software;
- shows the user how to operate the terminal and the shared mini hard copy devices;
- shows how to log on and how to log off;
- demonstrates how to access the library programs;
- introduces the concepts of files and file commands;
- how to use BASIC-PLUS;
- briefly describes the utilities, PIP, QUE, and EDIT;
- and lists the RSTS/E manuals.

Multimedia Show Access

Various methods of accessing the multimedia shows have been provided. In previous semesters academic staff members have been scheduled to take the shows to the classroom. In this situation a verbal introduction was provided, publications were passed out, the show was given, a "hands on" session provided, and questions were answered. After a couple of semesters, it was determined that this service was too extensive and too time-consuming. We were simply stretching a limited staff too thin considering that we had to maintain other services.

Currently, there are three ways of accessing the various multimedia shows:

- GROUP SHOWINGS IN THE WORKROOM - The Computer Center sets aside a special viewing area in one of the large user workrooms for the first four weeks of each semester. The instructor calls directly into the workroom to reserve time to bring his class to see particular multimedia shows. The consultant or lab assistant on duty starts the shows and may provide a "hands on" session if requested. During the "hands on" session, individuals are shown how to log on, how to log off and how to access a library program.
- CLASSROOM SHOWINGS - In this situation the instructor picks up the slides and cassette from the Computer Center and obtains the proper equipment (a Kodak Carousel slide projector and Wollensak slide sync cassette unit) from the audio-visual center. The instructor is also given appropriate publications to give to the students. He may also reserve a workroom "hands on" session to supplement the multimedia show. This service is available any time during the semester.
- CARAMATE SHOWINGS - Small single unit, slide carousel - cassette units are available in each workroom for individuals or small groups to view the multimedia shows. Users request a particular multimedia show from the on-duty consultant or lab assistant who shows them how to operate the Caramate unit. Caramate-viewed multimedia shows can be seen anytime the workrooms are open.

Multimedia Show Development

The original multimedia shows were, of course, costly to develop. The project leader developed content outline and scripts which went through many revisions and repeated approval. Frames corresponding to the script were described in detail and also subjected to review. The audio-visual department was used to shoot the pictures and reproduce the slides and cassettes. There were numerous delays and hangups for reshooting of the pictures and script revisions. It has been estimated that the early multimedia shows cost a man-year of time and $5000-$7000 a piece. The results were reasonable but not really effective. Essentially, the shows were too long, too detailed and contained too much information. After surveying faculty and students, new shows were produced. The second round of show production aimed at:

1) Reducing the overhead in both manpower and cost.
2) Providing multimedia shows that were crisp, clean and to the point.

In the second production, a special hourly consultant was hired (an ex-member of the staff). Having worked with the previous show producer, he was clearly aware of all the pitfalls and difficulties. This person has very special abilities: He is an excellant photographer and an excellent lecturer. This special consultant is shown the elements of the system and provided with available manuals. The major elements of the show and key topics are discussed with academic support personnel. Working from the "How To" publication and this background, the special consultant develops the show on his own. He writes the script, does the graphics, shoots the pictures for the slides and records the script. Development of the film and reproduction of the copies are all done commercially to reduce time delays. The results have improved a great deal. Over all production time has been reduced to about four weeks. The actual work time involved is 300 to 500 hours at a cost of $7.50/hour. The shows themselves have been reduced from 50 minutes in length to a maximum of 20 to 25 minutes. The time reduction came about because it was discovered that the longer shows presented too much detailed material and the attention span of the viewer could not be maintained after 20 minutes or so.

Media Considerations

The multimedia presentations are currently produced with 35mm color slides and audio cassettes. Initially other media was considered (i.e., film, filmstrips and video tape). Film, although producing a show of high quality, was too costly. Filmstrips

were used one semester but the copies of the shows were poor quality visually and operational problems with the filmstrips were persistent. Video tape is still under consideration but there exists a lack of easily accessible equipment in the university. Video monitors do well for small group showings but tend to be impractical for viewing by 30-40 students at a time. In the case of all three media - film, filmstrips, and video tape - the ease of updating the show is lost. With the use of slide/cassette production, "frames" of pictures or graphics can be reshot easily and inserted into the proper location in the carousel. The script is easily changed and retaped. A high quality stereo cassette unit is recommended to reduce extraneous noise. All the original taping of the scripts is done on a Kenwood stereo cassette model KX620 with Dolby.

RSTS/E PUBLICATIONS

One major service of the California State University, Northridge Computer Center is providing user-oriented publications and documentation. A limited academic support staff simply cannot answer all the questions users have about the various languages and systems. Four computer systems and well over a hundred vendor manuals simply overwhelms the novice user. Over a period of time, certain categories of Computer Center publications have been produced to assist the user.

Publications are available:

- to teach the user how to begin to use a system ("How To" publications);
- to condense information (the pocket-sized instants);
- to supplement existing documentation (auxiliary publications);
- to provide documentation when there isn't any (stand-alone documentation); and
- to provide information about library programs (Indexes and Sample Executions).

Publications For RSTS/E

A separate paper, USER-ORIENTED PUBLICATIONS: ON RSTS/E AND FOR RSTS/E[1] covers the California State University, Northridge RSTS/E publications in detail. The following publications are available for RSTS/E users:

- HOW TO USE RSTS/E TIMESHARING - This publication introduces the new user to RSTS/E timesharing. It contains an overview of the hardware, and the operating system. It tells the user how to use the terminal, how to log on and log off, how to create files and how to run programs in BASIC-PLUS and FORTRAN. This publication features a section on EDIT and briefly describes PIP and QUE. Manuals are listed and described. This particular publication is used in conjunction with the multimedia show, HOW TO USE RSTS/E TIMESHARING.
- BASIC-PLUS INSTANT - This is a pocket-sized publication containing an introduction, log on procedures, system commands, a special character section, the BASIC-PLUS language summary and an error message summary. It is available in booklet from or as a reference "card".
- RSTS/E FORTRAN INSTANT - A pocket-sized instant manual condensing key information from the RSTS/E FORTRAN manuals (RT-11/RSTS/E FORTRAN USERS GUIDE, PDP-11 FORTRAN Language Reference Manual, and RSTS/E FORTRAN IV Utilities Manual). It contains operating procedures, FORTRAN IV specific characteristics, FORTRAN language summary, library subroutines and error diagnostics. This is also available as an on-line publication.
- SYS SYSTEM FUNCTION CALLS - An on-line extraction of most of the non-priviledged SYS system function calls from the RSTS/E PROGRAMMING MANUAL.
- HOW TO USE RSTS/E EDIT - An on-line and hard copy publication written for the novice user and the slightly more advanced user. Features examples with line-by-line descriptions of what's going on.
- RUNOFF USER'S GUIDE - The DEC-provided on-line user's guide updated for the EDIT running on RSTS/E V6A. The high cost for the very thin RUNOFF manual motivated this effort.
- TIMESHARING LIBRARY PROGRAM STANDARDS - This on-line publication describes standards for timesharing library programs. The Standards are designed to make programs easy for the user to use. These Standards delineate a standard header (program name and description), standard output for questions to the user, standard processing of user input, format of self-contained program instructions and data entry format. The objective is to provide consistent, reliable, high quality programs for library users.
- RSTS/E TIMESHARING LIBRARY PROGRAM INDEX - Available on-line and in hard copy, this publication lists program names, category (BUSINESS, MATH, STAT, PHYSICS, etc.) and a short description of each program.

MICROFICHE RSTS/E MANUALS

The Problem:

The Computer Center has long provided vendor manuals for reference by users. Reference copies are available in the user workrooms and in the campus library. At times it was possible to provide free copies of vendor manuals for faculty and department libraries. Unfortunately, soaring paper and publication costs have limited the number of manuals that can be purchased. The free vendor manual service to faculty and departments was quickly terminated. Written justifications are nearly required to provide sets of manuals for the workrooms. Staff members are carefully queried as to their need for particular manuals. Management began to question the cost effectiveness of replacing manuals users ripped off from the workrooms. And there was always the time cost to be considered in updating both the workroom sets of manuals and individual sets of manuals.

The Solution:

The solution to the manual problem was to buy microfiche readers and to have nearly all vendor manuals microfiched. Reproduction permission was obtained

from our major vendors, Control Data Corporation and Digital Equipment Corporation. Use of these microfiched manuals is restricted to California State University and Colleges computer users.

Microfiche readers (approximately $200 each) were purchased for each workroom and each member of the Academic Applications group. A microfiched manual costs $.06 to $.25, compared to $7.00 to $21.00 for the hard copy version of the manual.

Each microfiche reader in the workrooms has an attached binder with all the microfiche in it. These are "reference" copies. Free microfiche are available for the asking at our user services desk.

Departments and individual users are now buying microfiche readers. The Computer Center is also considering the purchase of portable microfiche readers for users to "check out".

SPECIAL TOPIC SEMINARS

In addition to the multimedia shows, the various publications, and manual accessibility, the Computer Center offers another very important service - that of seminars. Seminars and workshops are offered on a variety of topics. When a new system or language becomes available, the Computer Center will offer seminars as soon as possible - often before any user documentation is readily available. Frequently seminars provide content and materials for user publications. Seminars and workshops are also provided on languages, systems, utilities and special packages that are considered "old stuff". Certain lectures are "recycled" periodically to introduce new users to "old" topics. The aim is to give a personalized lecture on how to use a certain facility. Seminars range from one two-hour session to four days of four-hour sessions depending upon the depth of the topic and the audience needs. Seminars are given by the various members of the instructional support staff and by our special "seminar" consultant.

SPECIAL INSTRUCTOR TUTORIALS

In order to better prepare instructors, the academic support staff offers individual tutorial sessions. Many instructors have come from other installations using a variety of computer systems. These special tutorial sessions acquaint the instructor with our particular computer systems. The special sessions are generally two sessions an hour or two long. On a one-to-one basis with a staff member, the instructor can obtain a brief but thorough view of the topic. We have specialized in EDIT, RUNOFF and DECAL sessions.

STUDENT AND FACULTY CONSULTING

All the other services - multimedia shows, publications, seminars, manual availability and special tutorials - require follow through. This is provided by the consulting service. Whenever workrooms are open, lab assistants or consultants are available to help the user. The main user workroom has staff members or student specialists that are on-duty as consultants (Monday through Friday, 8:00 AM to 5:00 PM). Other workrooms have student assistants, with limited consulting abilities, available nights and weekends. Specialized consulting from staff members is also available during the week.

Staff consultants and student assistants are "trained" on the use of the various systems by seminars and by having projects requiring them to learn about the system or language.

CONCLUSION

HOW TO USE RSTS/E: A USER-ORIENTED TRAINING PACKAGE has only highlighted a number of services conceived here as a package. Each of the components described, the multimedia shows, the publications, hard copy and microfiched copies of vendor manuals, special topic seminars, special instructor tutorials and consulting, are offered for the other systems available. The Academic Applications staff at CSU, Northridge supports large sized program libraries on the batch and timesharing systems. Additionally, we become involved with other aspects of instructional support such as introducing computer usage into non-computer using disciplines, offering high quality statistical consulting for faculty and graduate student research and converting large programming packages (such as SPSS, the Statistical Package for the Social Sciences).

Regardless of the multi-faceted activities, the Computer Center continually searches for ways of improving services. The components of the RSTS/E training package, like the ones for the other systems, are constantly being reworked after feedback from our users, our faculty and our own staff. Maintenance, is as important as development of support activities and training packages.

---

[1]Hayes, J.A.: USER-ORIENTED PUBLICATIONS: ON RSTS AND FOR RSTS, 1977 Fall DECUS U.S. Symposium.

DAY: Thursday, December 1
TIME: 10:45 AM - 11:15 AM
ROOM: DeAnza/Mesa

USER-ORIENTED PUBLICATIONS: ON RSTS/E AND FOR RSTS/E

J. A. Hayes, Academic Coordinator
Computer Center
California State University, Northridge
Northridge, California 91330

ABSTRACT

The Computer Center at California State University, Northridge strongly supports academic computing. The Academic Applications staff actively encourages instructional use of the various computer systems available. Underlying many kinds of instructional services is a strong documentation and publication system.

With the arrival of a 32-port PDP 11/45 running RSTS/E new possibilities for writing and compiling documentation and publications became apparent. The availability of both EDIT and RUNOFF on RSTS meant academic support personnel could have on-line documentation and publications that were both current and timely. Output can go to either an upper and lower case printer or to a DTC terminal to produce high quality, photo-ready copy. Library programs provide easy accessibility to users. This presentation will describe the documentation and publications on RSTS/E and for RSTS/E. Advantages and disadvantages of on-line documentation will also be discussed.

BACKGROUND

California State University, Northridge Computer Center has had a multi-system environment for a number of years. Because of having 3 to 4 computer systems it has become a necessity to produce user-oriented documentation* to introduce the users to the facilities of each system. Although the Computer Center is strongly instructionally-oriented, a substantial but not large instructional support staff exists to assist users. This group has written, compiled and published publications to supplement instructional computing activities. In many cases publications have been provided to alleviate many consulting questions. As we are all aware, the vendor documentation is always lacking in the introductory area. To fill the gap the novice has between no knowledge of a system and the basic vendor-provided manual, the Computer Center publishes a variety of user-oriented publications.

CATEGORIES OF PUBLICATIONS

Although not always clearly defined, there are basic categories of user-oriented documentation. For the purposes of this discussion, the following categories will suffice:

1. "How to" Publications
2. Instants
3. Auxiliary Publications
4. Stand-alone Documentation
5. Indexes and Sample Executions

A brief description of each of these categories will follow.

"How To" Publications

This category of publication normally provides an introductory view of a operating system. The user is given the essentials for accessing the system. A very brief delineation of the hardware is presented and the available system provided software is listed and described (i.e., languages and utilities). Installation software is described or, if lengthy, referenced. Manuals appropriate to the system are listed with ordering information. In the case of a "How To" for a timesharing system, the user is given information on logging on and logging off, file manipulation and utility use. The objective is to provide a starter set of information for the new user.

Instants

Normally pocket-sized, the "instant" publications provide the key elements and formats of a language or a system in very abbreviated form. The objective of the instant is to allow the user to see what's available and to see the format. Detailed information is directed toward the vendor manuals. The objective is to provide the user with a mini manual that is quick and easy to reference.

---

* "Documentation" will be used as a generic term for both program documentation and a more general category called publications. "Program" documentation, for these purposes, will refer to information about a particular program. Publications, on the other hand, will include a variety of topics about a system, or language, etc.

Auxiliary Publications

Auxiliary documentation is designed to provide the user with information that:

1. Clarifies the vendor documentation. The intent is to provide examples and to interpret.

2. Complements or supplements the vendor-produced documentation. This frequently encompasses installation specific features.

3. Is an extraction of "key" information.

Stand-alone Documentation

This category of documentation is generally the only source of information about a particular subject. Stand-alone documentation will be written when nothing else is available or if the documentation is extremely difficult or expensive to procure. The subject matter can be a language, a system, documentation for a program or such things as user-oriented Timesharing Library Program Standards.

Indexes and Sample Executions

Each computer system has its own application program library for instructional use. The supporting facility, in any case, provides an index of library programs. The application program library index includes the name of the program, category, and a short description of the program. While the index allows the user to see a list of available programs and the short description, a secondary level of presenting available library program information is also provided.

As many programs may provide what sounds like similar capabilities, the differences are demonstrated in a publication entitled, Timesharing Library Programs: Instructions and Sample Executions. In this publication the user gets to see an actual sample execution of a program. This sample run will almost always include the instructions or description of the program's capabilities. Novice users will see how to enter data; other users will want to note the unique features of similar programs. In hard copy form, this publication will also save the user valuable connect time on heavily loaded systems.

PUBLICATIONS ON RSTS/E

The Pros and Cons

With procurement of a 32 port PDP 11/45 supporting RSTS/E in October 1975, the Academic Applications group of the Computer Center discovered another tool and media for user-oriented publications. RSTS/E provided both a text editor and the word processor RUNOFF. The previous timesharing system, ITS (International Timesharing System) running on dual CDC 3170's had an easy-to-use editor but the campus had only 12 ports thus limiting production of on-line user documentation. The comparatively large number of ports on the local RSTS/E system along with RUNOFF lifted the inhibitions about doing on-line publications. Another important aspect of going on-line on RSTS/E was that programmer-consultants would learn the capabilities of both EDIT and RUNOFF thus enhancing their consulting abilities. In addition, student assistants cost less per hour to do the typing and make corrections.
So much for the advantages. The disadvantages have also become apparent as our RSTS/E system has become overloaded in the past year. Last spring an excess of 4000 students were competing for the 32 available ports making it difficult for the academic support staff to use RSTS/E for either program development or on-line documentation. While a special user program is available for accessing and allowing the user to drop off personal copies of a publication on the upper and lower case printer, soaring paper costs have limited this facility. Additional workloads coupled with a recent staff reduction has meant that some of the RSTS/E publications have been transferred to the secretarial staff for typing.

Access of On-Line Publications and Documentation

Although all on-line publications and documentation are available hard copy in the user workrooms, the user may also access the on-line materials by logging onto RSTS/E and calling up one of two programs. The first program, called INDEX (see Appendix A), pertains to the RSTS/E Application Program Library. As with all of our programs that meet Timesharing Library Standards, INDEX has self-contained instructions. The INDEX program will allow the user a number of different functions, for example:

A List of Program Names - All library programs can be listed or just categories (BUSINESS, CHEMISTRY, PHYSICS, MATH, STAT, etc.) of programs can be listed.

A List and Short Description of Each Program - Again, the user may choose to see all the programs, or just selected categories.

The list of program names, category and short description is available as a standard hard copy publication: RSTS/E Timesharing Library Program Index.

Sample Executions of Each Program - Each program on the library has an on-line sample execution. This permits the user to see an actual run of the program which includes the self-contained instruction and the user-program interaction. The user gets to see exactly what the output looks like.

Currently, for RSTS/E, the on-line access is the only means of obtaining the Sample Executions. For the previous central timesharing system and the current central system, Sample Executions has been available only in hard copy.

The second program, DOCUME (see Appendix B), was developed just for accessing other on-line documents. These on-line publications include program documentation where self-contained program instructions were substantial in size and publications on such information as a language or a system. The program DOCUME allows the user to see a list of on-line documents, allows him to select the one he wants and allows selection to the terminal or printer. (The terminal print feature is inhibited during heavily loaded times).

PUBLICATIONS FOR RSTS/E

This section will list and describe the current RSTS/E Version 6A publications:

The "How To" Publication

This is a 53 page booklet describing the essentials of using RSTS. Unlike the "How To's" for our other systems, the RSTS/E "How To" serves both the novice user and the slightly more advanced user. Design and format of this publication was extracted from PDP-11 PRIMER from University of California at Irvine.

HOW TO USE RSTS/E TIMESHARING - Features an excellent section on the Text Editor and a complete list of the character codes.

The following are key topics:

- RSTS/E TIMESHARING INTRODUCTION
- PPN - PROJECT - PROGRAMMER NUMBERS
- LOG ON
- LOG OUT
- FILES AND FILENAMES
- SYSTEM COMMANDS
- LIB - THE RSTS LIBRARY
- TTYSET, SET - TERMINAL CHARACTERISTICS
- PIP - THE PERIPHERAL INTERCHANGE PROGRAM
- QUE - LINE PRINTER REQUESTS
- EDIT - THE TEXT EDITOR
- BASIC-PLUS
- RSTS/E FORTRAN
- RSTS/E MACRO
- NEWS - SYSTEM NOTICES
- DIRECT - FILE DIRECTORY COMMAND
- CHARACTER CODES
- DIAGNOSTICS
- MANUALS

The Instants

The BASIC-PLUS INSTANT - With the arrival of the PDP 11/45, the BASIC-PLUS INSTANT was "instantly" produced. Since our users had been very used to an instant for timesharing, the initial version came about very rapidly. A brief introduction was written and summaries from the BASIC-PLUS manual were extracted. With a certain amount of arm-twisting our campus duplicating department reduced this to a nearly pocket sized publication. Like many new publications, the BASIC-PLUS INSTANT evolved into several forms. The Division of Information Systems, the central facility for the California State University and Colleges (California State University, Northridge is one of 19 campuses in this system) later produced the white card version (reminiscent of the IBM "green" card) and more recently, Northridge has added the BASIC-PLUS INSTANT to the on-line series. The content of the three forms remains basically the same:

- INTRODUCTION AND LOG ON PROCEDURES
- COMMAND SUMMARY
- SPECIAL CHARACTER SUMMARY
- BASIC-PLUS LANGUAGE SUMMARY
  - summary of variable types
  - summary of operators
  - summary of BASIC-PLUS statements
  - summary of functions
- ERROR MESSAGE SUMMARY

RSTS/E FORTRAN INSTANT - With the installation of RSTS/E FORTRAN came not one, but three more manuals. Again, there was a need for a quick reference document to bring the crucial elements of RSTS/E FORTRAN into one place. Part of the format was extracted from an RT-11 SYSTEM REFERENCE "card", the remainder is extractions from the FORTRAN manuals. The RSTS/E FORTRAN INSTANT was typed on-line by a student assistant and final copy was run off on a DTC impact print terminal at C.S.U., Northridge. The printing and distribution was handled by California State Polytechnic University, Pomona.

The contents of the RSTS/E FORTRAN INSTANT follow:

- OPERATING PROCEDURES
  - USING FORTRAN IV
  - RUNNING THE FORTRAN IV COMPILER
  - LINKING PROCEDURES
  - EXECUTION PROCEDURES
- FORTRAN IV SPECIFIC CHARACTERISTICS
- FORTRAN LANGUAGE SUMMARY
- LIBRARY SUBROUTINES
- FORTRAN IV ERROR DIAGNOSTICS

The RSTS/E FORTRAN INSTANT is available in a pocket sized version or available on-line.

Auxiliary Publications

For RSTS/E there are two publications which fall into the auxiliary publication category.

SYS SYSTEM FUNCTION CALLS - This is a direct extraction of most non-priviledged SYS system function calls from the RSTS/E Programming Manual. Ambitious students at San Francisco State University typed this one in on their own time. We transferred a copy via tape, did a little editing and installed it as part of our on-line documentation files.

HOW TO USE RSTS/E EDIT - At Northridge, one of the Computer Center's more ambitious student assistants became increasingly disatisfied with a couple of versions of the Text Editor Manual. He had worked extensively with EDIT on several projects and had concluded that the manual was not to be understood by the novice. For one thing, there were simply not enough examples with line-for-line explanations of what was going on. HOW TO USE RSTS EDIT was divided into two major sections, one for novice users and one for those who considered themselves more advanced. This publication includes a summary of EDIT commands, accesses to EDIT, explanations and examples of the majority of the EDIT commands, a brief explanation of the Dot and the Mark, an explanation of the EDIT page buffer, how to do large data insertions, use of the MACRO command, some commentary on EDIT error conditions and a summary of EDIT commands.

HOW TO USE RSTS/E EDIT is available on-line and as a hard copy publication.

Stand-Alone Documentation

Three major publications for RSTS exist in this category. Several minor handouts dealing with terminal and use of floppy disks are also available. The major publications include program documentation for one library program, the RUNOFF USER'S GUIDE and Timesharing Library Program Standards.

NORTON - extensive program documentation that was considered too large to be included in the program itself. NORTON gives planet and asteroid positions for any date from Jan. 1, 1900.

RUNOFF USER'S GUIDE - With our 5C version of RSTS/E came RUNOFF and an old version of the USER'S GUIDE on-line. Because of the large cost for such a small manual, the on-line version that came with the system was updated. Copies are available on-line or by special request.

TIMESHARING LIBRARY PROGRAM STANDARDS - California State University, Northridge served as the statewide or central timesharing facility for the 19 campuses of the California State University and Colleges system until December 1976. During the time we served as a statewide timesharing facility we increased the library from 100 to 300 programs. It was quickly realized that good user-oriented programs reduced the consulting load and the user's frustrations. Standards were developed to assure the user a certain level or quality of program. This publication delineates what a program should look like to the user; it standardizes instruction format and questions to the user. It is not a document on internal programming specification.

TIMESHARING LIBRARY PROGRAM STANDARDS is currently available as an on-line document.

Index and Sample Executions

RSTS/E TIMESHARING LIBRARY INDEX - Includes a list of program names, the category of the program, and a short description of the program. This publication is available on-line and in hard copy.

RSTS/E TIMESHARING LIBRARY PROGRAMS: INSTRUCTIONS AND SAMPLE EXECUTIONS - This was described earlier in the section titled "Access of On-Line Publications and Documentation". Each program has an actual sample run showing the instructions or program. Currently RSTS/E Library Program sample executions are available only on-line.

CONCLUSION

So what's this all about? USER-ORIENTED PUBLICATIONS ON RSTS/E AND FOR RSTS is one university's view on a service to instructional computing. Publications and program documentation is just one service to help the user help himself in an environment where there is 6 Academic Application staff members and a dozen or so student assistants to serve a potential computer user body of 9000 students and faculty members. We have essentially found that publications assist in keeping our consulting staff from getting an overload of questions. Publications from Northridge are basically installation independent so our publications can be used by the other 18 campuses in the system. Again, the aim is service, not only to our campus but to campuses that are unable to devote the time and effort to extensive publication activity.

APPENDIX A

SAMPLE EXECUTION OF THE PROGRAM INDEX

```
Ready

LIB INDEX

CSU NORTHRIDGE Application Library 10-Oct-77 07:40


I N D E X

Application Library Information

Do you want instructions? YES

INDEX allows users to selectively retrieve information about programs available on the application
library. The 'FUNCTION' query allows the user to specify the method by which the library information
is retrieved. The second query allows the user to specify a program name, category name, language
name, retrieval date, or output filename. An optional switch may be appended to the second response
to specify what information to print. A carriage return in response to any question will return the
legal responses.

Legal Functions (only the first three characters are necessary):

Category:    Lists programs by their category (i.e. Business, Math, etc.). The keyword
             'ALL' retrieves all programs. Switches are available.
.
.
.
Executions:  Allows printing of a sample execution for a program on the library. Switches
             cannot be used.

Exit:        Exits the program.

Help:        Prints these instructions.
.
.
.
Name:        Lists programs by their library name. The keyword'ALL' retrieves all programs. Switches are available.

Output:      Directs the output for the next function to a file. The file specification cannot contain a device specification or a PPN. The OUTPUT function is used in conjunction with any other function and is in effect for the NEXT FUNCTION ONLY. A CTRL/C will redirect output to the keyboard.

Legal switches are:

/N  Name:        Lists only the program name or names.

/S  Short list:  Lists program name, language, and category.

/L  Long List:   Lists program name, language, date, category and abstract.

Function? CATEGORY

Category? ALL/L

Biological sciences

Prog.   Lang.   Category

DRIFT   BASIC   Biological Sciences
Examines genetic drift with the Sewall-Wright effect
(change in gene frequency in a population due to
```

sampling error).

EPOP BASIC Biological Sciences
Demonstrates natural selection in evolution and how mutation rate, environment and heredity affect a population. ↑C

Function? EXECUTIONS

Program? MALAR

LIB MALAR

CSUC RSTS/E Application Library 23-Jun-76

M A L A R

HUNTINGTON II PROGRAM

MALARIA CONTROL SIMULATION

DO YOU WANT INSTRUCTIONS? YES

CURRENT CONDITIONS:
APPROX. POP. OF AREA: 100000
APPROX. NO. ILL WITH MALARIA: 25000
APPROX. NO. DEATHS/YR DUE TO MALARIA: 1000

TO CORRECT THIS SITUATION YOU CAN:
ISOLATE THOSE ILL IN QUARANTINE HOSPITALS
ADMINISTER DRUGS TO THOSE ILL
APPLY PESTICIDES TO KILL MOSQUITOES
GIVE PREVENTIVE DRUGS TO THOSE STILL HEALTHY

THESE ARE THE APPROX. COSTS:
FIELD HOSPITAL OF 20 BEDS: $2000 PER YEAR
DRUG TREATMENT FOR ILL: $2 PER PERSON FOR 1 YR.
FULL ANTI-MOSQUITO SPRAY: $75000 FOR 1 YEAR
WHEN USING DDT, OTHER SPRAYS HIGHER COST
PREVENTIVE DRUG EFFECTIVE 1 YR.:72 CENTS PER PERSON

YOU MAY USE THE PROGRAM EITHER WITH OR WITHOUT A BUDGET.

DO YOU WANT A LIST OF ASSUMPTIONS THE PROGRAM USES? YES

ASSUMPTIONS UNDER WHICH THE MODEL OPERATES

THE AREA UNDER TREATMENT HAS BEEN SUBJECT TO MALARIA FOR A LONG TIME AND THUS A FRACTION OF THE POPULATION IS ASSUMED TO HAVE DEVELOPED A SLIGHT RESISTANCE TO THE↑C

Function? EXIT

Ready

APPENDIX B

SAMPLE EXECUTION OF THE PROGRAM DOCUME

LIB DOCUME

CSU NORTHRIDGE Application Library 22-Jul-77 09:37

D O C U M E

ON-LINE DOCUMENTATION

THE DOCUMENTS CURRENTLY AVAILABLE ON-LINE ARE:

NUMBER	TITLE
1	RSTS/E RUNOFF
2	RSTS/E SYSTEM SYS FUNCTIONS
3	RSTS/E FORTRAN INSTANT
4	RSTS/E ASTRONOMICAL EPHEMERIS PROGRAM - NORTON
5	TIMESHARING LIBRARY PROGRAM STANDARDS
6	HOW TO USE RSTS/E EDIT
7	HOW TO USE NOS TIMESHARING

DO YOU WANT INSTRUCTIONS? YES

THIS PROGRAM PROVIDES THE USER WITH THE OPPORTUNITY TO OBTAIN DOCUMENTATION THAT IS CURRENTLY AVAILABLE ON-LINE. THESE DOCUMENTS DESCRIBE THE SYSTEM PROGRAMS (RSTS/E RUNOFF AND EDIT), THE AVAILABLE LANGUAGES (RSTS/E BASIC AND FORTRAN), THE SYSTEM SYS FUNCTIONS FOR RSTS/E, AND THE NETWORK OPERATING SYSTEM (NOS). THESE DOCUMENTS CAN BE DIRECTED TO THE RSTS/E LINE PRINTER, OR TO YOUR TERMINAL.

AFTER THE DOCUMENT HAS BEEN DISPLAYED, THE USER WILL BE GIVEN THE OPTION TO SEE ANOTHER DOCUMENT.

PLEASE TYPE THE NUMBER THAT IS ASSOCIATED WITH THE DESIRED DOCUMENT -- 6

YOU MAY OUTPUT THE DOCUMENTATION TO EITHER THE RSTS/E LINE PRINTER, OR YOUR TERMINAL. DO YOU WANT TO OUTPUT TO THE RSTS/E LINE PRINTER? YES

PLEASE TYPE NO MORE THAN THE FIRST SIX CHARACTERS OF YOUR LAST NAME SO THAT YOUR PRINTED DOCUMENT CAN BE IDENTIFIED AT THE COMPUTER CENTER -- SMITHR
.
.
.

YOUR REQUEST FOR A DOCUMENT TO BE PRINTED HAS BEEN PROCESSED. YOU MAY PICK UP THE PRINTOUT IN A FEW HOURS AT THE USER'S SERVICES DESK AT ENGR. 121. THE PRINTOUT WILL HAVE A HEADER CONSISTING OF YOUR ACCOUNT NUMBER AND NAME PRINTED ON THE FIRST PAGE.

DO YOU WISH TO OUTPUT ANOTHER DOCUMENT? NO

Ready

Day: Thursday, Dec. 1
Time: 2:30-3:00
Room: Sunset

DYNPRI Design Considerations

DYNPRI is a program that enables the user to have some control over the performance of his RSTS system. The RSTS internal scheduler provides basic timesharing operation by time slicing job execution and overlapping one jobs execution with another jobs I/O and swapping. DYNPRI is not so much a fixed program as it is a concept of external scheduling control by a running job. External control of the scheduler is possible by setting each jobs external scheduling parameters - priority and run burst. DYNPRI periodically accesses monitor tables, determines some characteristics about the other jobs, and sets the external scheduling parameters according to a particular control algorithm. The characteristics determined about the jobs and the control algorithm used vary among the different versions of DYNPRI. An accurate determination of the relative effectiveness of various control algorithms requires a measure of system performance given various configuration dependencies. The development of a RSTS Performance Monitor is a separate topic to be addressed in the near future. This discussion will cover the philosophy of external control and some control algorithms.

D A T A T R I E V E - 1 1

D E C Q U E R Y / R E P O R T S Y S T E M

F O R

P D P - 1 1

JAMES STARKEY
DIGITAL EQUIPMENT CORPORATION
DECEMBER 1, 1977

DATATRIEVE-11
FUNCTIONALITY OVERVIEW

- INTERFACES WITH RMS-11 FILES
  - SEQUENTIAL
  - INDEXED
  - RELATIVE
- PERFORMS SELECTIVE RETRIEVAL
  - KEYED FIELDS
  - NON-KEYED FIELDS
- PERFORMS FILE MAINTENANCE FUNCTIONS
  - CREATES NEW RECORD
  - MODIFIES EXISTING RECORDS
  - DELETES OLD RECORDS
- GENERATES REPORTS
  - MULTIPLE CONTROL BREAK
  - AUTOMATIC TOTALING
  - AUTOMATIC FORMATTING
  - REPORT/PAGE HEADERS AND TRAILERS

DATATRIEVE-11
PRODUCT GOALS

1. SIMPLICITY OF USE
   - SIMPLE, ENGLISH-LIKE SYNTAX
   - FORGIVING PARSER
   - QUALITY DIAGNOSTICS
   - ELIMINATION OF SPECIAL CASES (E.G. KEYED/NON-KEYED RETRIVAL)
2. BROAD FUNCTIONALITY
   - SELECTIVE RETRIEVAL
   - FULL UPDATING
   - REPORT GENERATION
   - EXTENSIVE FORMATTING CAPABILITIES
3. INTERFACE WITH EXISTING FILES
   - COBOL
   - FORTRAN
   - BASIC+2
   - DIBOL

DATATRIEVE-11
POTENTIAL USERS

- NON-TECHNICALLY ORIENTED END USER
  - CLERK
  - SECRETARIES
  - MANAGERS
- TECHNICAL STAFF
  - APPLICATION PROGRAMMERS
  - DATA ADMINISTRATOR

DATATRIEVE-11
POTENTIAL APPLICATIONS

- AD HOC QUERIES
- ONE SHOT REPORTS
- DIRECT END USER ACCESS TO EXISTING FILES
- COMPLETE SYSTEM FOR SIMPLE APPLICATIONS

DATATRIEVE-11
SOFTWARE REQUIREMENTS

- OPERATING SYSTEMS
  - IAS (VERSION 2.0)
  - RSX-11M (VERSION 3.0)
  - RSTS/E (VERSION 6B)
- BUNDLED WITH RMS-11K

DATATRIEVE-11
HARDWARE REQUIREMENTS

- 64KB OF USER MEMORY
- 600 + BLOCK OF DISK STORE EXCLUSIVE OF USER FILES
- EXTENDED INSTRUCTION SET
- FLOATING POINT PROCESSOR IF FLOATING DATA TYPES ARE USED

DATATRIEVE-11 CONCEPTS
DATA DICTIONARY

- CREATED BY SUPPLIED UTILITY
- MAINTAINED BY DATATRIEVE
- STORES RECORD, DOMAIN, PROCEDURE DEFINITIONS
- SHARED AMONG USERS
- SOFTWARE PROTECTION AT LOGICAL RECORD LEVEL

DATATRIEVE-11 CONCEPTS
RECORD DEFINITION

- RECORD TYPE DEFINED BY DEFINE RECORD STATEMENT
- RECORD DEFINITION LANGUAGE IS UPWARDS COMPATIBLE FROM COBOL
- DEFINITION IS VERIFIED FOR SYNTAX, STORED IN DATA DICTIONARY
- DOES NOT INCLUDE FILE NAME, FILE TYPE, OR FILE STRUCTURE

DATATRIEVE-11 CONCEPTS
RECORD DEFINITION EXAMPLE

```
DEFINE RECORD YACHT USING
Ø1 BOAT.
   Ø3  TYPE.
      Ø6  MANUFACTURER PIC X(1Ø)
          QUERY-NAME IS BUILDER.
      Ø6  MODEL PIC X(1Ø)
   Ø3  SPECIFICATIONS
       QUERY-NAME SPECS.
      Ø6  RIG PIC X(6).
      Ø6  LENGTH-OVER-ALL PIC XXX
          QUERY-NAME IS LOA.
      Ø6  DISPLACEMENT PIC 99999
          QUERY-HEADER IS "WEIGHT"
          EDIT-STRING IS ZZ,ZZ9
          QUERY-NAME IS DISP.
      Ø6  BEAM PIC 99.
      Ø6  PRICE PIC 99999
          EDIT-STRING IS $$$,$$$.
```

DATATRIEVE-11 CONCEPTS

COBOL RECORD DEFINITION EXTENSIONS

- QUERY-NAME IS NAME
  - OPTIONAL
  - ALLOWS SHORTHAND NAME
- EDIT-STRING IS PRICTURE-EDIT-STRING
  - OPTIONAL
  - PROVIDES DEFAULT PRING FORMAT FOR FIELD
- QUERY-HEADER IS HEADER-STRING
  - OPTIONAL
  - PROVIDES DEFAULT PRING COLUMN HEADER FOR FIELD

DATATRIEVE-11 CONCEPTS
RECORD DEFINITION EXAMPLE

```
Ø6  MANUFACTURER PIC X(1Ø)
    QUERY-NAME IS BUILDER.
```

DATATRIEVE-11 CONCEPTS
RECORD DEFINITION EXAMPLE

```
Ø6  PRICE PIC 99999
    EDIT-STRING IS $$$,$$$.
```

DATATRIEVE-11 CONCEPTS
RECORD DEFINITION EXAMPLE

```
Ø6  DISPLACEMENT PIC 99999
    QUERY-HEADER IS "WEIGHT"
    EDIT-STRING IS ZZ,ZZ9
    QUERY-NAME IS DISP.
```

DATATRIEVE-11 CONCEPTS

DOMAIN

- CREATED BY DEFINE DOMAIN STATEMENT
- SPECIFIES RECORD TYPE AND FILE NAME
- DEFINITION IS VERIFIED FOR SYNTAX, STORED IN DATA DICTIONARY
- DOES NOT INCLUDE FILE TYPE OR FILE STRUCTURE
- ACCESS (READ, WRITE, MODIFY, EXTEND) CAN BE CONTROLLED BY PASSOWRD AND/OR VIC PROTECTION
- DOMAIN NAME IS "HANDLE" FOR FILE IN DATATRIEVE

DATATRIEVE-11 CONCEPTS
DOMAIN DEFINITION EXAMPLE

```
DEFINE DOMAIN YACHTS USING YACHT ON DB1: [305,305] YACHT.DAT;
```

DATATRIEVE-11 CONCEPTS
COLLECTION

- SUBSET OF A DOMAIN (SINGLE RECORD TYPE)
- ESTABLISHED BY EXECUTION OF DATATRIEVE FIND STATEMENT
- CAN BE RE-ORDERED BY DATATRIEVE SORT STATEMENT
- EXISTS ONLY FOR DURATION OF DATATRIEVE SESSIONS

DATATRIEVE-11 CONCEPTS
PROCEDURE

- CREATED BY DEFINE PROCEDURE STATEMENT
- MAY BE A CLAUSE, STATEMENT, OR GROUP OF STATEMENTS
- STORED IN DATA DICTIONARY

DATATRIEVE-11 CONCEPTS
PROCEDURE DEFINITION EXAMPLE

```
DEFINE PROCEDURE STORE-YACHT
  STORE YACHTS VERIFY USING
    BEGIN
      IF BEAM EQ 0 THEN ABORT "BAD BEAM"
      IF DISP EQ 0 THEN ABORT "BAD DISPLACEMENT"
      IF NOT LOA BETWEEN 20 AND 60 THEN ABORT
          "BAD LENGTH"
      PRINT
      DISPLAY "CONFIRM WITH Y IF OK"
      IF *.CONFIRMATION NE "Y" THEN ABORT
          "STORE ABORTED"
    END
END-PROCEDURE;
```

DATATRIEVE-11 LANGUAGE OVERVIEW
READY STATEMENT

- MAKE DOMAIN AVAILABLE FOR ACCESS
- MAY SPECIFY ACCESS TYPE
  - READ - RETRIEVAL ONLY
  - MODIFY - RETRIEVAL, VALUE MODIFICATION
  - WRITE - RETRIEVAL, FULL UPDATE
  - EXTEND - STORE ONLY
  - DEFAULT IS READ
- MAY SPECIFY ACCESS MODIFIES
  - SHARED - OTHERS MAY READ OR WRITE
  - PROTECTED - OTHER MAY READ
  - EXCLUSIVE - OTHER USERS LOCKED OUT

  DEFAULT IS PROTECTED
- PERFORMS BINDING OF DOMAIN NAME, RECORD NAME, FIELD NAMES, AND FILE STRUCTURE
- PERFORMS ACCESS PERMISSION, REJECTS UNAUTHORIZE ACCESS
- EXAMPLE

READY YACHTS

DATATRIEVE-11 LANGUAGE OVERVIEW
FIND STATEMENT

- ESTABLISHES COLLECTION
- MAY BE SELECTIVE
- OPERATORS PROVIDED:
  - EQUAL
  - NOT-EQUAL
  - GREATER-THAN
  - LESS-THAN
  - GREATER-EQUAL
  - LESS-EQUAL
  - BETWEEN
  - AND
  - OR
  - NOT
- EXAMPLE:

FIND YACTS WITH PRICE BETWEEN

10000 AND 20000

[24 RECORDS FOUND]

DATATRIEVE-11 LANGUAGE OVERVIEW
FIND CURRENT STATEMENT

- REFINES PREVIOUSLY ESTABLISHED COLLECTION
- ESTABLISHES NEW COLLECTION
- EXAMPLE

FIND CURRENT WITH RIG EQ "SLOOP", "KETCH"

[22 RECORDS] FOUND

DATATRIEVE-11 LANGUAGE OVERVIEW

SORT STATEMENT

- RE-ORDER COLLECTION
- DOES NOT RE-ORDER ORIGINAL FILE
- MULTIPLE SORT KEYS ALLOWED
- MAY BE ASCENDING, DESCENDING, OR MIXED
- EXAMPLE

  SORT BY LOA, DISPLACEMENT

DATATRIEVE-11 LANGUAGE OVERVIEW

PRINT ALL STATEMENT

- PRINTS ALL RECORDS OF COLLECTION
- DEFAULT IS TO TERMINAL
- OUTPUT MAY BE RE-DIRECTED
- DEFAULT PRINTS ALL FIELDS, EXPLICIT FIELDS MAY BE SPECIFIED
- COLUMN HEADER IS AUTOMATICALLY GENERATED

CONTINUED

DATATRIEVE-11 LANGUAGE OVERVIEW

PRINT ALL STATEMENT (CONT.)

- EXAMPLES

  PRINT ALL ON LP:

  PRINT ALL TYPE, PRICE

MANUFACTURER	MODEL	PRICE
WINDPOWER	IMPLUSE	$3,500
CAPE DORY	TYPHOON	$4,295
VENTURE	21	$2,823
VENTURE	222	$3,564
ISLANDER	BAHAMA	$6,500
SALT	19	$6,590
CAPE DORY	25	$8,995
IRWIN	25	$10,950
AMERICAN	26	$9,895
ALBIN	79	$17,900
TANZER	26	$11,750

DATATRIEVE-11 LANGUAGE OVERVIEW

SELECT STATEMENT

- SELECTS A SINGLE RECORD IN COLLECTION FOR SPECIAL PROCESSING
- SELECTION MAY BE:
  - FIRST
  - NEXT
  - LAST
  - N-TH
- EXAMPLES:

  SELECT FIRST

  :

  SELECT NEXT

DATATRIEVE-11 LANGUAGE OVERVIEW

PRINT STATEMENT

- LIKE PRINT ALL, BUT PRINTS SINGLE RECORD

DATATRIEVE-11 LANGUAGE OVERVIEW

MODIFY STATEMENT

- MODIFY CHANGES VALUES IN SELECTED RECORD
- MODIFY ALL CHANGES VALUES IN ALL RECORDS IN COLLECTION
- ANY NUMBER OF FIELDS MAY BE MODIFIED
- PROMPTS FOR NEW VALUES
- REQUIRES MODIFY OR WRITE ACCESS TO DOMAIN
- EXAMPLE:

```
MODIFY PRICE
PLEASE SUPPLY VALUE FOR PRICE: 3500
```

DATATRIEVE-11 LANGUAGE OVERVIEW

STORE STATEMENT

- CREATES NEW RECORD IN FILE
- REQUIRES WRITE ACCESS TO DOMAIN
- DEFAULT FORM PROMPTS FOR ALL FIELDS
- OPTIONAL VALVE VERIFICATION MAY BE SPECIFIED
- EXAMPLE:

```
STORE YACHTS
PLEASE SUPPLY VALUE FOR MANUFACTURER: HINKLEI
PLEASE SUPPLY VALUE FOR MODEL: BERMUDA 40
PLEASE SUPPLY VALUE FOR RIG: YAWL
PLEASE SUPPLY VALUE FOR LENGTH-OVERALL: 40
PLEASE SUPPLY VALUE FOR DISPLACEMENT: 22000
PLEASE SUPPLY VALUE FOR BEAM: 12
PLEASE SUPPLY VALUE FOR PRICE: 82000
```

DATATRIEVE-11 LANGUAGE OVERVIEW

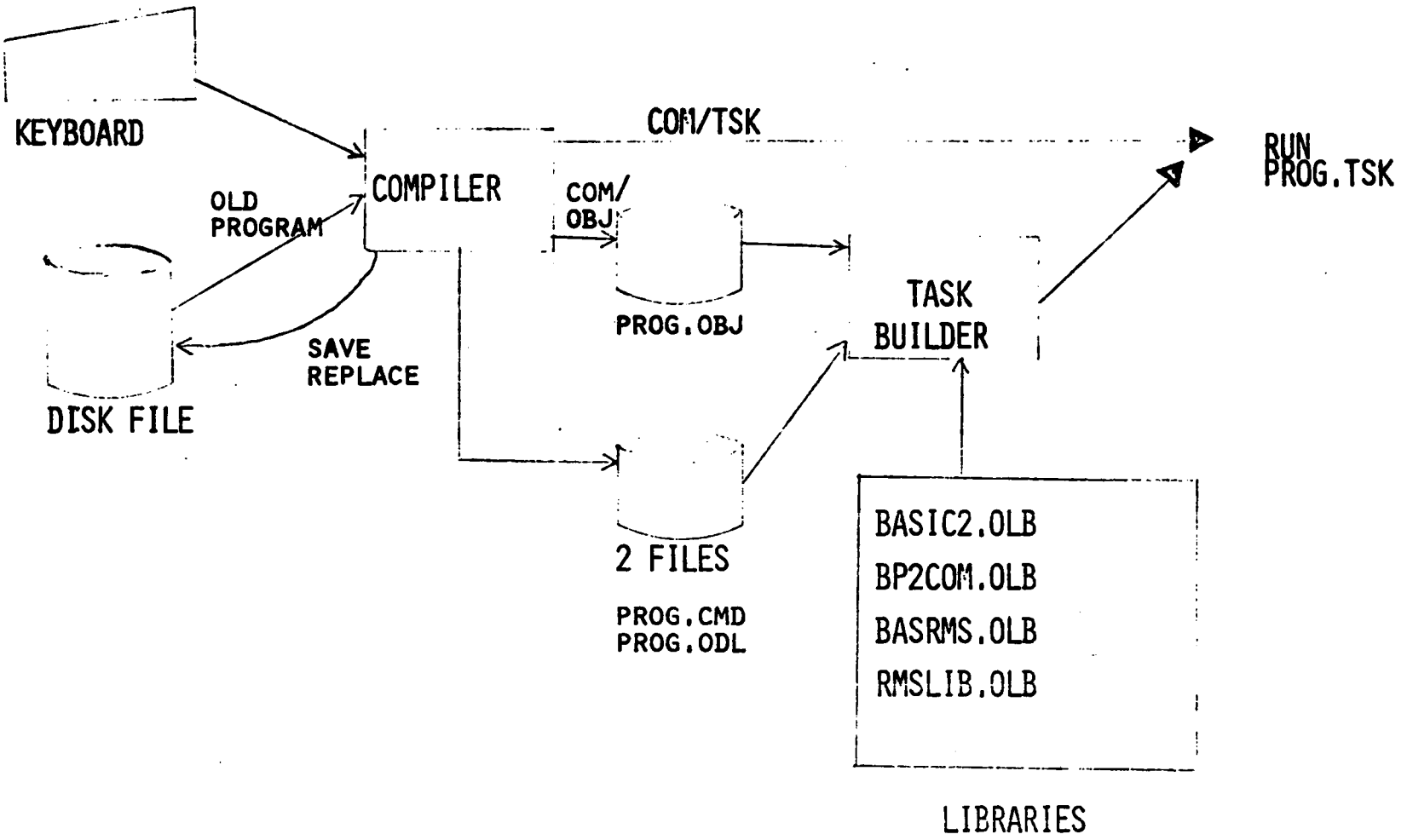
REPORT WRITER

- INVOKED FOR WITHIN DATATRIEVE
- MAY ACCESS COLLECTIONS
- FOLLOWING MAY BE SPECIFIED:
  - REPORT NAME
  - MULTIPLE CONTROL BREAKS
  - AUTOMATIC STATISTICAL FUNCTIONS
    MAX, MIN, AVERAGE, TOTAL, COUNT
  - AUTOMATIC FORMATTING
  - PAGE AND LINE LIMITATIONS
  - OUTPUT DEVICE
  - OUTPUT PAGE DIMENSIONS
  - SUMMARY OR DETAIL REPORT
  - AUTOMATIC OR MANUAL DATE
  - PAGE NUMBERING
  - PAGE TOTALS
  - REPORT TOTALS

B A S I C + 2

KERBEY ALTMANN
DIGITAL EQUIPMENT CORPORATION

GENERATING A BASIC PLUS 2 TASK



BP2COM RUNTIME SYSTEM

1. IS 4 K WORDS IN LENGTH

2. INCLUDES:
   - (A) KEYBOARD MONITOR
     1. CCL COMMANDS
     2. CAN BE USER DEFAULT RTS
     3. RECOGNIZES:
        - (A) ASSIGN
        - (B) ATTACH
        - (C) BYE
        - (D) DEASSIGN
        - (E) DISMOUNT
        - (F) HELLO
        - (G) LOGIN
        - (H) MOUNT
        - (I) REASSIGN
        - (J) RUN
   - (B) RSX EMULATOR
     - (1) ALL RSX FUNCTIONALITY IN RSX.RTS
     - (2) EXTRA FEATURES
   - (C) POST MORTEM DUMP
     DUMPS ALL OF USER CORE TO A FILE
     "PMDØXX.PMD" IN BINARY

3. HAS A VERY MINIMUM OF ROUTINES

BASIC 2 RUNTIME SYSTEM

1. IS 16K WORDS IN LENGTH

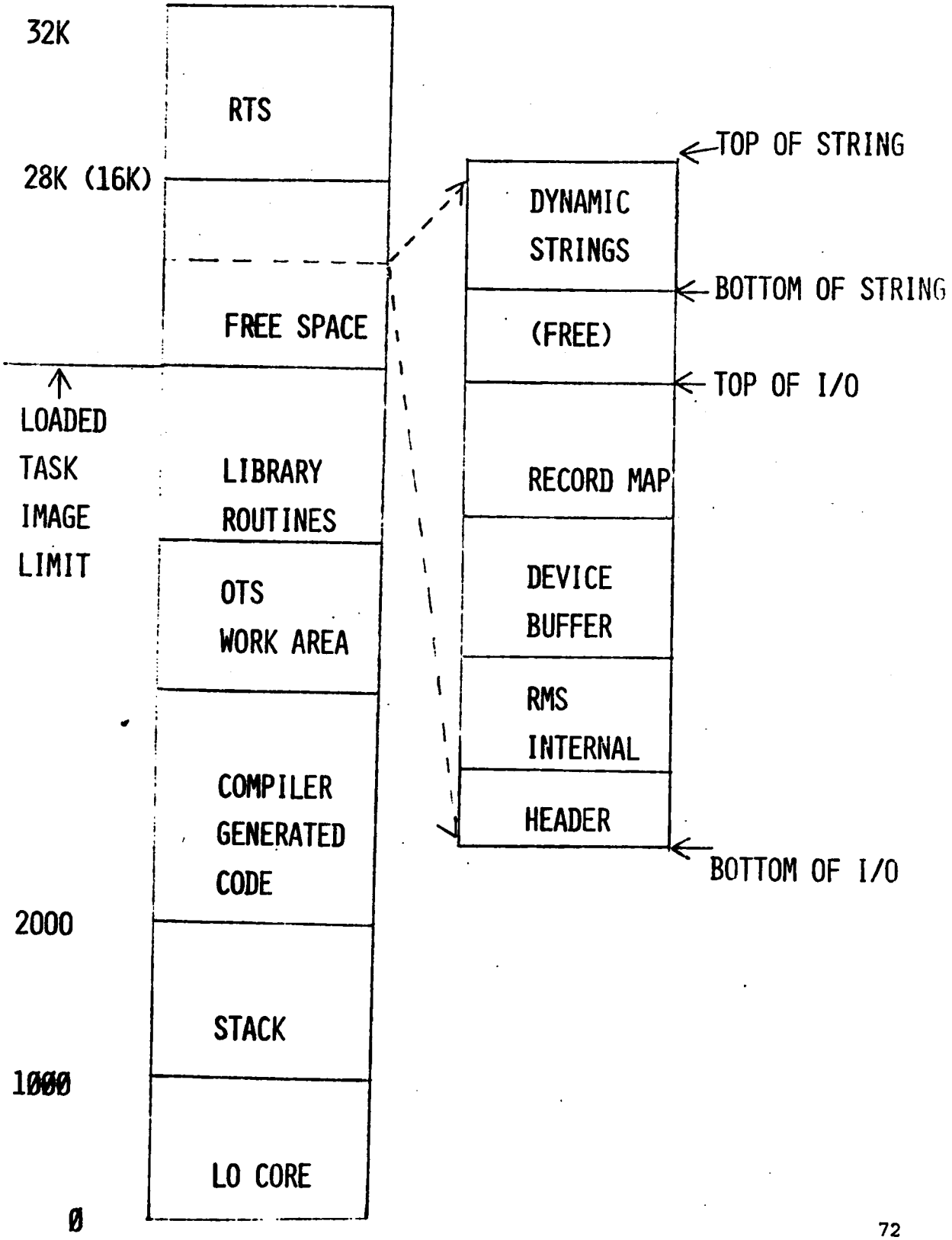
2. INCLUDES:
   - (A) RSX EMULATOR
   - (B) POST MORTEM DUMP

3. HAS ALL ROUTINES EXCEPT:
   - (A) PARAMETER PASSING
   - (B) INDEX KEY PROCESSING
   - (C) RMS INTERFACE
   - (D) DEBUGGER (EIS + FIS)

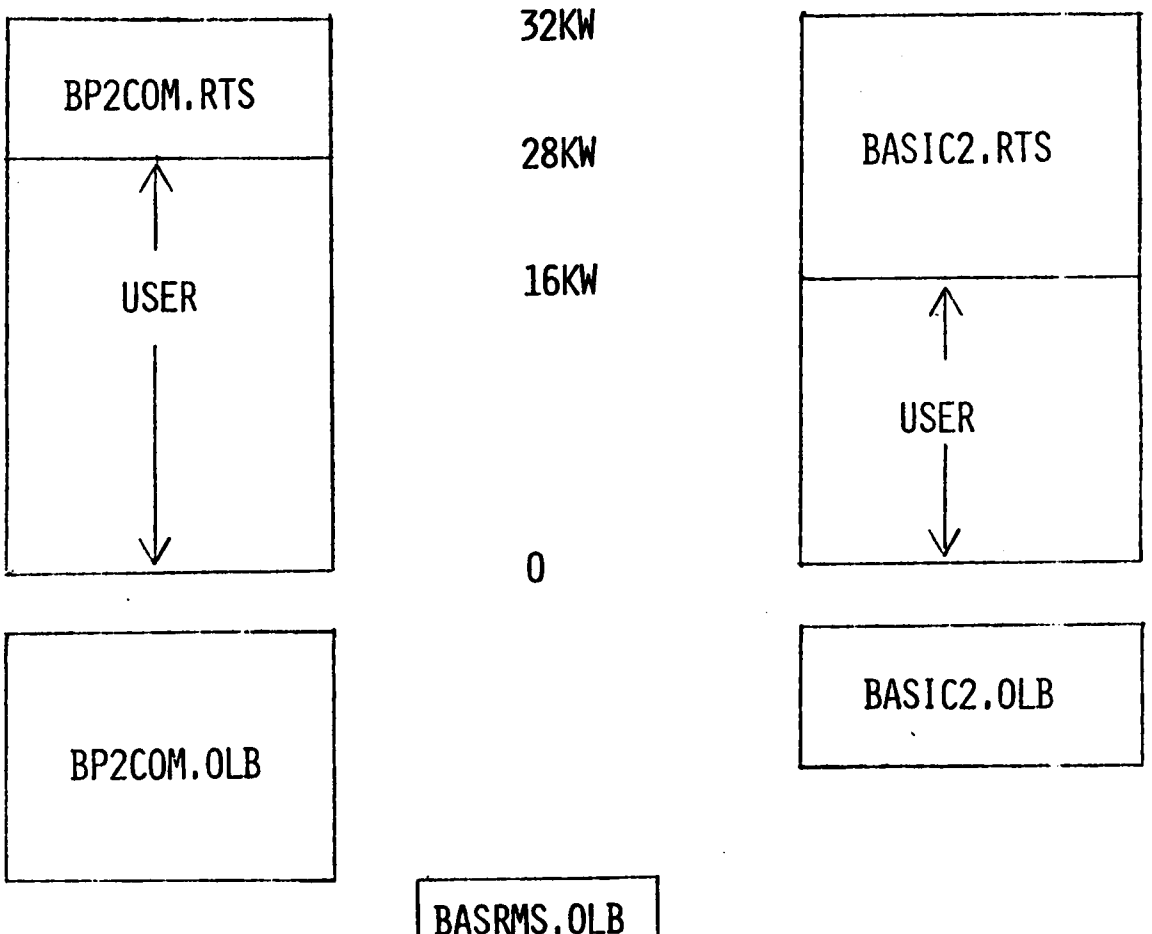
4. IS NOT A KEYBOARD MONITOR

5. COMES IN THREE VERSIONS:
   - (A) EIS - NO HARDWARE FPU
   - (B) FIS - PDP-11/40 ONLY
   - (C) FPU - PDP-11/34, 11/45 AND 11/70 ONLY

RUN TIME ENVIRONMENT



CORE USAGE


32KW
28KW
16KW
0
BP2COM.RTS
USER
BP2COM.OLB
BASIC2.RTS
USER
BASIC2.OLB
BASRMS.OLB
RMSLIB.OLB

DIFFERENCES BETWEEN

BASIC-PLUS	BASIC PLUS 2
A. SYNTAX	
1. CONTUATION LINE	
(A) LINE FEED	(A) NO-
(B) AMPERSAND -CR (EXTEND MODE)	(B) ONLY WAY
2. STATEMENT SEPARATORS	
(A) COLON(:)	(A) NO-
(B) BACKSLASH (\)	(B) ONLY WAY
3. COMMENTS	
(A) ONLY STATEMENT (REM OR EXCLAMATION (!)	(A) YES
(B) END OF STATEMENT	(B) YES
(C) BEGIN WITH (!) END WITH (& -CR)	(C) YES
(D) N/A	(D) EMBEDDED IN STATEMENT BETWEEN A PAIR OF (!)
4. CHAIN STATEMENT	
(A) CHAIN "<NAMES>" <LINE #>	(A) CHAIN "<NAME>" LINE <LINE #> KEYWORD LINE ADDED
5. RANDOM NUMBERS	
(A) A = RND (X%)	(A) A = RND NO ARGUMENT

DIFFERENCES

B. ENHANCEMENTS

1. LINPUT
   (A) ACCEPTS THE ENTIRE LINE WITHOUT THE TERMINATORS.
   (B) CANNOT USE A STRING PROMPT

2. PRINT USING
   (A) JUSTIFIED STRINGS
      (1) LEFT "'LLL"
      (2) RIGHT - "'RRR"
      (3) CENTER - "'CCC"
   (B) EXTENDED STRINGS "E"
   (C) SINGLE QUOTE (') NOW A FORMAT CHARACTER
   (D) EXCLAMATION MARK NO LONGER SINGLE FIELD
   (E) MUST NOT HAVE COMMA (,) BETWEEN CHANNEL NUMBER AND "USING"

3. IF - THEN - ELSE
   (A) CAN HAVE ANY NUMBER OF STATEMENTS BETWEEN IF AND THEN AND AFTER ELSE
   (B) MUST HAVE A STATEMENT WITH A LINE NUMBER AFTER LAST STATEMENT IN ELSE CLAUSE.

4. CALL STATEMENT
   (A) CAN CALL
      (1) SEPARATELY COMPILED BASIC PLUS 2 SUBPROGRAMS
      (2) MACRO SUBROUTINES
      (3) SYSTEM SUPPLIED SUBROUTINES
         (A) DECNET
         (B) PLOT55
         (C) GRAPHICS PACKAGES
         (D) SORT-11V2

   (B) TWO TYPES OF CALL
      (1) PLAIN - CALL
         (A) ARRAY HEADERS
         (B) STRING LEADERS
      (2) DEC STANDARD - CALL BY REF ADDRESSES OF ALL PARAMETERS

5. COMMON STATEMENTS
   (A) GLOBAL VARIABLES SHARED BETWEEN SUBPROGRAM UNITS
   (B) EXACTLY LIKE FORTRAN
   (C) NAMED PSECTS
   (D) RSX-11M GLOBAL SHARED COMMONS

6. GREATLY ENHANCED OPEN STATEMENT

7. DEF VS DEF*
   (A) DEF* FOR COMPATIBILITY WITH BASIC PLUS
   (B) DEF NOT YET IMPLEMENTED

C. DIFFERING RESULTS

1. AMBIGUOUS CONSTANTS
   (A) IN BASIC PLUS, USAGE DEPENDS ON IF THERE IS INTERGER TO LEFT
   (B) IN BASIC PLUS 2, ALWAYS FLOATING POINT

2. EMBEDDED BLANKS IN DATA STATEMENTS
   (A) IN BASIC PLUS, THEY ARE ELIMINATED UNLESS ENCLOSED IN QUOTES
   (B) IN BASIC PLUS 2, ALWAYS RETAINED

3. STRING HANDLING
   (A) IN BASIC PLUS, EQUATING TWO STRINGS SIMPLY SETS UP A SECOND HEADER POINTING INTO A COMMON DATA AREA
   (B) IN BASIC PLUS 2, A COMPLETELY SEPARATE COPY IS CREATED

4. END STATEMENT
   (A) BASIC PLUS
      (1) REQUIRED
      (2) DOES NOT WRITE OUT BUFFERS

   (B) (1) NOT REQUIRED, INSERTED BY COMPILER
      (2) PERFORMS IMPLICIT CLOSE ON ALL FILES; WRITES OUT BUFFERS.

5. MULTIPLE ASSIGNMENTS
   (A) BASIC PLUS EVALUATES RIGHT TO LEFT
   (B) BASIC PLUS 2 EVALUATES LEFT TO RIGHT

C. 6. CHAIN STATEMENT

(A) BASIC PLUS

(1) CHANNELS CLOSE

(2) BUFFERS NOT WRITTEN OUT

(3) SOURCE PROGRAM COMPILED BEFORE EXECUTION, IF NECESSARY

(B) BASIC PLUS 2

(1) ALL BUFFERS WRITTEN OUT

(2) CHANNELS CLOSED

(3) FILE MUST BE IN EXECUTABLE FOR (I.E. .TSK)

(4) EXTENSIONS ARE LOOKED FOR IN ORDER OF RTS

7. TWO DIMENSION SUBSCRIPT CHECKING

(A) BASIC PLUS

(1) FIRST SUBSCRIPT CHECK FOR IN RANGE

(2) SECOND CAN "WRAP AROUND"

(B) BASIC PLUS 2 - BOTH SUBSCRIPTS CHECKED SEPARATELY

8. UNTERMINATED STRINGS

(A) BASIC PLUS - NOT NECESSARY TO TERMINATE STRING LITERAL

(B) BASIC PLUS 2 REQUIRES, OR AN ERROR RESULT

D. DIFFERENT FUNCTIONS

1. POS (X%) - CCPOS (X%)
   (A) BASIC PLUS USED POS TO RETURN TERMINAL PRINT POSITION
   (B) BASIC PLUS 2 USES CCPOS

2. TAB (X%)
   BASIC PLUS 2 - THIS FUNCTION CAN BE USED TO RETURN A STRING OF X% BLANKS

3. INSTR (N%, M$, S$) - POS (M$, S$, N%)
   (A) BASIC PLUS USED INSTR TO FIND A SUBSTRING IN A MAIN STRING
   (B) BASIC PLUS 2:
      (1) ALLOWS INSTR FOR COMPATABILITY
      (2) PREFERRED USAGE IS THE POS FUNCTION

4. MID (M$, B%, L%) - SEG $ (M$, B%, E%)
   (A) BASIC PLUS - MID TAKES STRING M$ AND RETURNS A SUBSTRING L% LONG STARTING AT POSITION B%
   (B) BASIC PLUS 2:
      (1) ALLOWS MID FOR COMPATABILITY
      (2) SEG$ SPECIFIES ENDING POSITION (E%) INSTEAD OF LENGTH

5. NUM$ - STR$
   (A) BASIC PLUS - NUM$ ACCEPTS A NUMBER AND OUTPUTS A STRING IDENTICAL TO A PRINT
   (B) BASIC PLUS 2:
      (1) ALLOWS NUM$ + NUM1$ FOR COMPATABILITY
      (2) STR$ IS IDENTICAL

6. CVT$$ - EDIT$
   (A) BASIC PLUS - CVT$$
   (B) BASIC PLUS 2 - EDIT$
      IDENTICAL

7. TRM$
   BASIC PLUS 2 - EQUIVALENT TO EDIT$ (128%) STRIP TRAILING BLANKS + NULLS

E NOT YET IMPLEMENTED

1. FOREIGN BUFFERS - BUT CAN USE THE SAME MAP IN OPEN STATEMENTS

   EXAMPLE:

```
10 OPEN A$ FOR INPUT AS FILE #1%, &
      MAP ABC
20 OPEN B$ FOR OUTPUT AS FILE #2%, &
      MAP ABC
30 MAP (ABC) A$ = 512
40 GET #1%\PUT #2%
50 GO TO 40
```

2. PRINT WITH RECORD

NEW FILE OPEN

```
10 OPEN A$ [FOR INPUT ] AS FILE #1%
           [    OUTPUT]

     , ORGANIZATION  VIRTUAL
                     SEQUENTIAL [FIXED   ]
                     RELATIVE   [VARIABLE]
                     INDEXED    [STREAM  ]

     , ACCESS [READ  ]  , ALLOW [READ  ]
              [WRITE ]         [MODIFY]
              [MODIFY]         [NONE  ]
              [APPEND]         [WRITE ]
              [SCRATCH]

     , MAP BUF  , RECORDSIZE A%

     , BLOCKSIZE N%, BUCKETSIZE N%

     , FILESIZE N%, CLUSTERSIZE N%

     , TEMPORARY , CONTIGUOUS

     , NO SPAN

     , PRIMARY   [KEY] K1$ [DUPLICATES]

     , ALTERNATE [KEY] K2$ [DUPLICATES]
                           [CHANGES  ]

20 MAP (BUF) K1$ = 6, K2$ = 6, REC$ = 68
```

RECORD OPERATIONS

```
FIND        #1%          , RECORD B%

GET         #1%          , KEY #2%  EQ  A$
                                   GT
                                   GE

PUT         #1%          , COUNT B%

UPDATE      #1%          , RECORD A%

DELETE      #1%

RESTORE     #1%          , KEY #A%

SCRATCH     #1%
```

KNOWN RESTRICTIONS

1. RESTORE NOT SUPPORTED FOR TERMINAL FORMAT FILES.

2. NO SINGLE DATA ELEMENT MAY BE GREATER THAN 16 KW.

3. CCPOS
   (A) NOT CORRECT FOR ESC.
   (B) NOT COUNT ↑ CHARACTER.
   (C) IF KEYBOARD OPENED ON MORE THAN ONE CHANNEL, CCPOS NOT SUM.

4. CONTROL C TO COMPILE WILL BOMB COMPILER.

5. ALLOW MODIFY FOR INDEXED OR '
   RELATIVE FILES NOT PERMIT EXTENDING FILE.

6. RMS FILE SPECIFICATION USED ON
   (A) ALL RMS FILES
   (B) OBJ OUTPUT OF COMPILER

   MUST BE IN FORM IF

   <PHYSICAL DEVICE> : [PPN] <FILE NAME>

   NOT ALLOWED
   1. LOGICAL DEVICES
   2. PROTECTION CODES
   3. FILE SWITCHES
   4. PPN AT END OF FILE NAME

Overview of the RSTS System

Speaker, Robert Daley

Digital Equipment Corp.

Maynard, Mass.

This session is oriented toward members of the DECUS community who desire an overview of the RSTS/E system, its capabilities, its languages and product set, and its potential application areas. Today's RSTS system has evolved from RSTS-11, a highly effective and easy to use, BASIC-oriented timesharing system, into a general purpose computing system capable of supporting a broad spectrum of application areas. The RSTS system will be presented both in terms of its technical capabilities (e.g., its product set, language and utility support, communications and network support, etc.) and in terms of areas of application (e.g., in business, education, science, etc.). The session is intended to leave the audience with a solid technical overview of the total capabilities and potential of the RSTS system and the general direction Digital has established for the future. The session will also provide a useful background to many of the other RSTS sessions at DECUS.

RSTS/E -- THE SYSTEM TODAY

- A MULTI-FUNCTION SYSTEM
- A MULTI-LANGUAGE SYSTEM
- A BROAD BASE OF PRODUCTS AND FUNCTIONALITY
- A SYSTEM OF PROVEN QUALITY
- A BROAD BASE OF USERS

HISTORY RSTS-11 RSTS/E

- 1972-1972 RSTS-11
  - SINGLE LANGUAGE, BASIC PLUS
  - TIME SHARING ONLY
  - STAND-ALONE SYSGEN
- 1973-1974 RSTS/E (VERSION 5)
  - LARGER CONFIGURATION SUPPORT (MEMORY MANAGEMENT)
  - BATCH
  - LINE PRINTER SPOOLING
  - BACKUP
  - 2780
- 1975-1977 RSTS/E (VERSION 6)
  - MULTI LANGUAGE SUPPORT (COBOL, FORTRAN)
  - RMS-11, RMS-11K
  - EXTENSIVE NEW HARDWARE SUPPORT
  - EXTENSIVE MONITOR ENHANCEMENTS
- CURRENT ACTIVITIES (TO BE SUPPLIED AT DECUS)

RSTS/E, A MULTIFUNCTION SYSTEM

- TIME SHARING
  - INSTRUCTIONAL
  - PROGRAM DEVELOPMENT
- INTERACTIVE APPLICATIONS
  - SINGLE AND MULTI-TERMINAL APPLICATIONS
  - INTER-JOB COMMUNICATION
  - TRANSACTION PROCESSING
  - FIELD-LEVEL TERMINAL (CRT) INPUT MODE
- BATCH PROCESSING
  - MULTIPLE BATCH STREAMS
  - NO HUMAN INTERVENTION REQUIRED
  - ON LINE SUBMITTAL, NO CARD READER NEEDED
- LINE PRINTER SPOOLING
  - RSTS PRINTERS
  - 2780 SPOOLING
- DISTRIBUTED PROCESSING

RSTS/E, A MULTI-LANGUAGE SYSTEM

- BASIC PLUS
  - POWERFUL SUPERSET OF BASIC LANGUAGE
  - INTERPRETIVE
  - IMMEDIATE MODE
  - EASY TO USE
- BASIC PLUS II
  - SUPERSET OF BASIC PLUS LANGUAGE
  - COMPILED OBJECT CODE (OVERLAYS, SUBPROGRAMS)
  - EXTENSIVE DATA MANAGEMENT FACILITIES
- COBOL
  - 1974 ANSI STANDARD
  - MULTI-TERMINAL SUPPORT ON RSTS/E
  - SEQUENTIAL, RELATIVE AND MULTI-KEY ISAM SUPPORT
  - COMPILES 5000-6000 LINE PROGRAMS
  - SUBPROGRAMS, OVERLAYS
- FORTRAN IV
  - FULL ANSI STANDARD
  - FAST COMPILATION/EXECUTION
  - SUBPROGRAMS, OVERLAYS
- OTHER LANGUAGES
  - RPG-II
  - APL
  - MACRO
  - DIBOL/DECFORM (CTS-500 ONLY)

RSTS/E -- PRODUCTS AND FUNCTIONALITY

- BASE PRODUCT SET
  - BASIC PLUS
  - RMS-11, SEQUENTIAL AND RELATIVE ACCESS
  - FULL SET OF COMMANDS AND UTILITIES
  - EXTENSIVE SYSTEM MANAGMENT FACILITIES
  - RSTS/E MULTI-FUNCTION MONITOR
  - MULTI-TERMINAL SUPPORT
  - INTER-JOB COMMUNICATION
- AVAILABLE PRODUCTS FROM DIGITAL
  - ADDITIONAL LANGUAGES (COBOL, FORTRAN, ETC.)
  - MULTI-KEY ISAM (RMS-11K)
  - FAST SORT
  - 2780
- BROAD BASE OF PRODUCTS FROM USERS, DECUS AND SOFTWARE HOUSES

RSTS/E -- A SYSTEM OF PROVEN QUALITY

- "EASE OF USE" (SYSTEM AND DOCUMENTATION)
  - FOR TIME-SHARING AND BATCH USERS
  - FOR APPLICATION DEVELOPERS
  - FOR THE SYSTEM MANAGER
- PERFORMANCE
  - OPTIMIZED AND OVERLAPPED DISK OPERATIONS
  - SHARED RUN TIME SYSTEMS
  - TERMINAL RESPONSE TIME
  - SYSTEM THROUGHPUT
  - NEWER RELEASES WILL CONTINUE TO OUTPERFORM OLDER RELEASES
- RELIABILITY/MAINTAINABILITY
  - ON-LINE ERROR LOGGING
  - AUTOMATIC SYSTEM UPDATE (AUTO PATCH)
  - TIME TESTED IN DIVERSE ENVIRONMENTS

RSTS/E -- A BROAD USER BASE

- OVER 3000 RSTS/E USERS
  - MANUFACTURING
  - DISTRIBUTION
  - INSURANCE
  - BANKING
  - TRANSPORTATION
  - EDUCATION (ADMINISTRATION, INSTRUCTION, RESEARCH)
  - MANY OTHER APPLICATIONS

- THE ADDED VALUE, A LARGE USER BASE
  - RICH SOFTWARE BASE
    - USER INTERCHANGE
    - DECUS LIBRARY
    - SOFTWARE HOUSES, OEM'S
  - RICH SOURCE OF FEEDBACK TO DIGITAL
  - YOU KEEP US HONEST

RSTS/E -- FOCUS ON THE FUTURE

- DISTRIBUTED PROCESSING
- LANGUAGES AND PRODUCT SET
- PERFORMANCE, RELIABILITY
- FILE AND DATA MANAGEMENT
- APPLICATION "TOOLS"
- CONTINUED HIGH LEVEL OF SUPPORT AND COMMITMENT

RSTS/E ACCOUNTING AND INTERNAL STRUCTURE
TASK FORCE OPEN MEETING

Chairman: Bill Sconce

Day: Tuesday, November 29
Time: 4:45 - 5:15 PM
Room: Senate/Committee

General: We will be asking Digital only for nuclear, basic functionality, and documentation of standard interfaces with the system. It is presumed that specific needs of various installations will vary widely, so that a system manager will still have to do final tailoring. Our object here is to outline areas which may require operating system support, or which are indeed very widely needed.

JOBLOG

An ERRLOG-like facility, whereby accounting data can be sent from the Monitor (or from system programs) to be logged in a disk file, or used by local system-management programs. Some possible message types:

1. Job initiation (normal or spawn)
2. LOGIN
3. RTS change
4. Device assign and de-assign
5. File open and close
6. Disk access count at CLOSE
7. SPOOL: pages printed
8. LOGOUT
9. Job teardown

Job Data (implications for LOGIN, SYSTAT, etc)

1. Operator Name
2. Session identification

MFD Structure

1. User data area for each PPN
2. Supporting SYS calls
3. Priority, coremax, run burst at LOGIN
4. PPN Name
5. Pointer to financial records
6. LOGIN message
7. PPN expiration date

LOGIN Program (implications for MFD structure)

1. Standard interface in LOGIN.BAS for user code
2. Limiting PPN's to certain KB's, times, days
3. Auto speed recognition
4. Query to set terminal characteristics
5. User-to-user "mailbox"
6. Access by an "operators'" password
7. Automatic CHAIN
8. Mnemonics for PPN's
9. One-time password

NOTICE.TXT Facility

1. A push-down message history
2. Supporting program, such as NEWS
3. Short notice not killed by slash

LOGOUT

1. Option to change password
2. Echo "Killing" and "Saving"
3. Don't kill write-protected files
4. List dollar charges, and by device

Other

1. Password Changer program
2. MONEY - accounting by device
3. MONEY - option for total CPU usage
4. REACT - create an ACCT.SYS equivalent from the existing MFD
5. REACT - optional input files
6. REACT - ACCT.SYS separate for different disks

RSTS/E WISHLIST: SELECTIVE DISCUSSION

Chairman: Bill Sconce

Day: Wednesday, Nov 30
Time: 10:45 - 11:45
Room: Golden West

Tentative Agenda:

1. Maximum filesize of 65535
   ..probably the oldest friend on the wishlist

2. Stabilized/Small Versions of RSTS/E

3. Relief for [1,2]
   BUILD: variable PPN's
   Directory roots in core for !, %, &, @, #, etc.

4. Software Maintenance Hotline

5. Improved control of File Processor scheduling
   Priority balancing; user control facilities

6. _____

7. _____

8. _____

→ Reviewing the agenda will be the first order of business

Other Candidates:

+ Program Logic Manual: RSTS/E Internals

+ 1200 Baud Half Duplex support

+ Two or more levels of privilege

+ ROLLOUT-ROLLIN; Keyboard Funnel

+ Suggestions from the floor

A note: included on anyone's list of RSTS/E priorities would be the continued support and/or enhancement of interpretive Basic-Plus, the BACKUP package, system CUSP's, and the RSTS/E accounting structure. The reason they were not included for the agenda is that they are receiving other coverage; Basic-Plus in the Basic-Plus II sub-SIG meeting (Weds am). BACKUP and CUSP's by their respective SIG Task Forces, and accounting structure in a session of its own (Tues am).

1. Maximum Filesize of 65535 (29)

Question. The maximum size of a RSTS/E file is 65535 512-byte blocks. Many applications require a larger total amount of data; is there to be any help ?

Discussion. This question was posed at the Fall '75 DECUS in Los Angeles. The response from DEC at that time was that enlarging the file maximum would be impractical, and it went out in the following Chan 13 as an "Answer". The furor isn't over, however; judging from the unsolicited response at Atlanta, Las Vegas, and Boston, this one will have to remain on the wishlist.

---

2. Stabilized/Small Versions of RSTS/E (126)

Remarks. There IS a small, stabilized RSTS around - Version 4: it will run on an 11/20, but it's not a supported product. While V4 is an extreme case, there would be general benefits to be obtained from having available a stabilized/small version of RSTS/E, among them less maintenance and lower core requirements, Not having to generate new systems periodically would reduce overhead - and patching activity would eventually subside, too. Less core means less hardware expense, obviously, but main storage limitations are beginning to be a major factor for 11/45's and down - 128K gets tighter with each new release. One user writes, "Some respect for users of smaller machines, please! IE, 11/34's which must run the current system to be supported..."

There are major questions, of course. What facilities should be included? How big, exactly? Modularity, so size and performance can be traded off? And what version of RSTS/E should be used as a base?

3a. [1,2] is Too Big (243)

Question. Users have several times voiced concern that the system library is so large as to be cumbersome, and still grows with each release. Not only does it become harder to manage, but system response is degraded as well.

Discussion. There is, of course, good reason for having a library which all users can access easily. However, many programs in [1,2] are never directly called by any timesharing user: the spooler structure; LOGIN, LOGOUT, etc; programs invoked by CCL's. Others, being seldom used (SHUTUP, for instance) could really go in any PPN, if they could be forced to the end of the UFD.

Further Comments. With 6B, there is technical support for hooking CCL's to programs in any PPN. Also, users have mentioned organizing libraries by Run-Time System.

3b. BUILD: Variable PPN's (255)

Discussion. If everything didn't HAVE to go into [1,2], the system library could be kept much leaner. Of course, to build a system with multiple libraries, the BUILD procedure would have to be able to resolve explicit references in the source code; something like variable symbols would be required.

Further comments. COSAP, a statistical package for RSTS/E, does exactly this kind of thing. Any PPN can be used, and the BUILD procedure hooks everything up correctly.

3c. Directory Roots in Core (351)

Question. The disk addresses for the beginning of the directories of [1,2] and [self] are kept in Monitor core and job-related core respectively, to speed up access. What about other PPN's ?

Discussion. If a certain PPN is likely to be used by either the user community or by a certain job, knowing where to start looking for its User File Directory can eliminate references to the Master File Directory, and improve performance. PPN's which the Monitor could remember for all jobs include the libraries, plus "service" PPN's in which might reside CCL programs, LOGIN, etc. Additional PPN's which might be remembered in the job area include [*,0] (pound sign) and the assignable PPN (@).

- - - - - - - - - - - - - - - -

4. Software Maintenance Hotline (43)

Question. How about a phone number in Maynard which a user could call when in trouble with software ?

Discussion. This suggestion was made in December 1975, at DECUS Los Angeles, and has come in for repeated discussion since then. Digital has at least given consideration to the idea. There would need to be some controls, obviously - phone interruptions could adversely affect normal SPR response, or even dilute development time. Several users have suggested means by which the service could be limited in such a way as to keep it cost-effective.

- - - - - - - - - - - - - - - -

5. FIP Priorities (39)

Question. Would it be possible for the priority of heavy disk users to be dynamically adjusted in such a way that overall system performance isn't degraded by one or two users ?

Discussion. With the exception of putting users having a priority of >0 first, the RSTS/E File Processor (FIP) currently works on a simple first-in, first-out basis. This means that a job having light CPU requirements but using much disk I/O can slow the system down significantly, particularly when SY: is also used for swapping. Unfortunately, using $PRIOR (or a dynamic priority balancer either) to adjust priority has little effect when a job doesn't need much CPU time. What's needed is for FIP to help balance how many disk accesses each job gets.

- - - - - - - - - - - - - - - -

\+ Program Logic Manual (58)

Question. It has been rumored that an internals manual was going to prepared for publication in the Version 6B time frame. ??

Discussion. The current situation (November 1977) seems to be that no PLM project has been funded. Further, the PLM would have been intended for internal DEC use; there never was a decision to offer it for sale. However, neither the question (nor the possible project, evidently) is dead yet.

We have some difficult questions in this area. On the one hand, publication and sale of specifications for internal interfaces, monitor calls, etc. could be construed as a commitment for support; this would be appropriate only at a time when RSTS/E is finally ready to be "frozen". We may or may not want to hasten that day.

more →

PLM, continued...

On the other hand, selling a manual without support commitment would be an unusual procedure for a computer manufacturer. We may want to do some thinking about signed waivers, etc.

.. MACRO under RSTS/E (18)

Question. Will use of assembler language (MACRO) be supported under RSTS/E ?

Discussion. There seem to be no such plans beyond the use of MACRO subroutines to support Fortran programs. MACRO use in this context is somewhat limited, in that it supports neither I/O nor monitor calls.

User's comment, March 1977: want MACRO with I/O capabilities.

Further comments. This question is related to the problems presented by publication of a Program Logic Manual: implied support, and additionally, compatibility across systems. Especially where MACRO I/O is concerned, creating software to run in native RSTS mode (without using the RMS interface) results in programs which will run only in a single environment. Transportablility to another operating system, such as RSX-11M, would be lost - and so would transportability to possible future hardware.

These questions are difficult, partly because they involve balancing short-term vs long-term considerations. It's always tempting to view today's configuration as stable - but it's often not easy to be objective. Further, there are of course applications which, once designed for a time-sharing environment, no one would want to transport to 11M.

.. RTS Documentation (1)

Question. RSTS/E now supports multiple run-time systems (also called object-time systems, RTS, or OTS). Basic-Plus, Fortran, and COBOL each have one; new languages will have theirs; and SORT-11 executes using pure assembly-language code in an RTS. If used heavily enough, any program can be enhanced by implementation in pure code. The TECO editor works this way; so do several OEM packages -- but they were written by people having more than the average insight into the workings of RSTS/E, to say the least. How about publishing specs on how an RTS must be constructed ?

Discussion. There are no current plans to make these specifications widely available. Several things should be considered. Publishing any documentation involves non-trivial expenses, which are difficult to spread equitably if only a few installations will use it. Publication would not "freeze" or guarantee that the specs could not change in later releases, forcing updates of any user-written RTS's. Finally, customers desiring to use their own RTS facilities would probably have to waive DEC software support for that part of their systems.

\+ 1200-Baud Half Duplex Support (341)

Question. Many users would like to make use of half-duplex facilities.

Discussion. This feature has at least been in the implementation plan at one point, but its schedule has been slipping.

+ ROLLOUT-ROLLIN (61)

Question. Has this ever happened to you? You're debugging a program; it has STOP's sticking out all over it, you've modified several variables, and so on. The next thing you need is to look at some part of the input file, but naturally don't want to lose the status of your current job. If you work in the computer center, you might just go to another terminal, but even that is an interruption. This facility would let you save your porcupined program, current status and all (ROLLOUT), run PIP or whatever, then restore the debugging job exactly where it was (ROLLIN).

Discussion. Digital says this one would be VERY difficult to provide. A remote user can get almost the same effect by hanging up on a dial-up line (letting his debugging job detach), logging in a new job for the PIP operation, and finally dialing up again to ATTACH to the original job. Some installations run an AUTOKILL program that won't let a detached user job live long enough for this to work, however.

User comment. "We have ROLLOUT working in our shop now; ROLLIN is proving a little tougher."

.. KB: Funnel Utility (236)

Question. The idea would be to allow a single real terminal (probably a CRT) to simulate several virtual terminals. An operator could then control several simultaneous production jobs, in an interactive fashion not possible through BATCH. Alternatively, an interactive user could gain a kind of "look-aside" capability, and interleave several dialogues. It is, after all, RSTS' facility for interactive work, and the resulting people-productivity, that makes a RSTS shop so unusually effective.

- - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - -

RSTS/E NEW USERS' SESSION

Chairman: Dr. John Scogin III

Day: Wednesday, Nov 30
Time: 3:15 - 5:15
Room: Golden West

A session for new kids on the block, and
for old kids on the block, too

- Discussion Groups -

1. What everyone else already knows about RSTS/E.
2. Small system operations
3. Large system operations
4. How to deal with DEC sales
5. How to deal with DEC Field Service
6. Problems in the academic environment
7. Use accounting, and billing
8. Tailoring CUSP's for special purposes
9. How to track down a hardware problem before calling Field Service
10. To be announced

Presentor: Tom Barnard
11, West Liegh
Bingley, Yorkshire

Day: Tuesday Nov 29th
Time: 08:30 - 09:00
Room: Golden West Room

RPL11M

HISTORY

We have developed a compiler for use at Airfix Industries that runs from three to ten times faster than BASIC+ in one quarter to one half the true core occupancy, and in which programs can be written up to five times faster than in BASIC+, and can use virtual core with variable length pages.

This success has only been achieved through the application of experience gained previously in developing a similar compiler on IBM360 and ICL1900 computers and after having a second try on RSTS/E PDP11. This is a story of some length and complexity and is to do with doing things differently and more effectively, and in being persistant over a long period of time (ten years!).

In 1963 Tom Barnard was working as a programmer at Morgan Crucible Co in Wandsworth, London, England and became obsessed with the repetitiveness of report programming that he had to tackle as part of day-to-day work in the programming shop. He set himself the objective of automating the process of report writing with syntax usable directly by users. He developed clear ideas of how the problems of defining reports to a computer could be handled, including multiple control breaks, different types of heading, difference between character transfer information and numeric totaling information, all for which syntax was easy to define that users could undertand, but until reading an IBM pamphlet on decision tables, could not see how ordinary mortals could be brought to state the logical complexities of record selection and calculations on data shown in reports.

Any problem that can be stated in IF DO or in IF GOTO logic can be solved more clearly with decision tables. Generally few people need a training course to use decision tables as grasping the principles of decision tables takes minutes and graduates in the technique can begin to read international airline timetables correctly. A simple example is given of travelling on London underground routes from Leicester Square.

Destination ?						
Buckingham Palace	YES	-	-	-	-	NO
Houses of Parliment	-	YES	-	-	-	NO
Regents Park	-	-	YES	-	-	NO
St Pauls Cathedral	-	-	-	YES	-	NO
10 Downing Street	-	-	-	-	YES	NO
Route						
Piccadilly line to Piccadilly Circus	-	-	X	-	-	-
Picadilly line to Holborn	-	-	-	X	-	-
Northern line to Charing Cross	X	X	-	-	-	-
Change Tubes	X	X	X	X	X	-
Bakerloo line to Regents Park	-	-	X	-	-	-
Central line to St Pauls	-	-	-	X	-	-
District/Circle to Wesminster	-	X	-	-	X	-
District/Circle line to St James's park	X	-	-	-	-	-
Ask a Policeman	-	-	-	-	-	X

Tom left Morgans and developed NITA (Nineteen Hundred Tabulator) for the ICL 1900 in 1967 and convinced the then fledgling British National Computing Centre of the sense of his cause and the effectiveness of NITA which then contained:

- Report Formatting Syntax
- Record Selection and Calculations by Decision Tables
- A Core Sort for Totals Only Reports

and he joined the NCC who started selling NITA.

At the NCC Tom developed an enhanced NITA called FILETAB with a free form decision table syntax and got this working on ICL1900 and IBM360. The British oil giant BP became interested and funded the development of a Univac 1100 version, and a Honeywell version came to be as well.

Tom left the NCC in 1972 to emigrate to New Zealand and we will pick up the RSTS/E part of the story later....NOW READ ON!

In 1967 Alan Browne was a fledgling programmer at British Aircraft Corportation in Stevenage, Hertfordshire, having joined that company from being a line foreman and factory methods engineer at a big furniture manufacturing company. British Aircraft Corporation had had Will Stevens of Urwick Diebold (the London consultants) to implement a decision table method of specifying systems, and as a result Alan was indulging in the masochistic task of taking systems specifications beautifully and concisely stated in decision tables and writing them as programs with IF GOTO logic in COBOL on an IBM360 - (PERFORM was then forbidden for core saving reasons). He had had an initial baptism of fire in trying this in RPG and nearly lost his job! [Kit Grindley of Urwicks is now pursuing the ultimate objective of "Systematics" a method of specifying systems directly to a computer and has lectured on this subject in England with Gerry Weinberg of New York State University (of some fame!)].

Alan went on to work briefly for the Gulf Oil Corporation where he used RPG in a generally COBOL environment to do things quickly. He states categorically that RPG is at least twice as fast to write as COBOL and up to five times as fast, given that the RPG listing is not more than three pages long. Every extra page after three doubles the length of time to write the program in RPG. This is because the syntax is so bad to state solutions in, and because it takes so much time to comprehend a line when reading, what indicators are on or off, and whether the programmer or the files set them on or off, or not etc. However, RPG can produce fast results (even if they don't run so fast!)

At the end of 1968 Alan joined Hoskyns in London, who may be known in the USA for having sold their Modular Applications Systems (manufacturing company application programs in COBOL) to UNIVAC for free issue to customers and for now being part of Martin Marietta computer operations. At Hoskyns Alan used FILETAB, and with a decision table systems specification background from British Aircraft

Corporation, got results far beyond anything he had achieved with RPG.

From Hoskyns in 1970 Alan joined Airfix Products, a client of Hoskyns, who were located about a mile away in Wandsworth from where Tom Barnard had started at Morgan Crucible in 1963. Alan joined Airfix as Data Processing Manager, and found his Chief Programmer Joe Lamberty to be a NITA advocate having come to Airfix from Morgans. Alan bought FILETAB from the NCC and under Joe Lamberty's direction the whole Airfix programming operation soon started using FILETAB in a major way, and made several significant innovations:

- No Systems-Analyst versus Programmer distinctions;- programmers started meeting users and systems analysts started writing programs

- No systems specifications of the type that are conventionally handed from analyst to programmer:- Users were given the option of saying what they thought they might like, and then the right to change the requirement when the sight of real data showed the need for this: user and programmer could solve most problems in a few days.

- No documentation, other than the listing and file layouts:- decision table programming produces uniform solutions - i.e. all exception conditions are self evident and different people produce the same decision tables as solutions to the same problem. A decision table program is self documenting to a degree that is amazing to anyone who has ever had all ten fingers marking pages in a COBOL listing (and writing with a pen gripped in his/her teeth!).

The limitations of FILETAB, which had been designed as a report writer, were working against these innovations, and when Tom Barnard came back to England from New Zealand in 1973 Airfix set up a joint venture with him to develop a paging (virtual core) decision table compiler for the 1900 to contain FILETAB as a subset of its syntax. This was developed in a year and used to create an Airfix Optical Mark Reading System for field order taking by fifty sales representatives on one thousand products sold to ten thousand outlets - the virtual core facility was used in translating the many sheet types.

This compiler was called DTPL (Decision Table Programming Language) and was subsequently licenced to the NCC, who have further developed it and called it FTL6 - the NCC FILETAB package is now one of the few European software products to sell over 500 copies.

In 1973 Airfix bought a PDP11/40 with RSTS/E to develop on-line applications and Tom Barnard started work with them on a RSTS/E version of DTPL, to be called RPL (Report Programming Language).

Being in a rush the major error was made of going for a preprocessor approach which like all other BASIC+ program generators was to live in the shadow of 16K for ever (though BASIC+2 might help - when it gets across the Atlantic!). The preprocessor took six months to develop and works fine for report production - for one and two input file reports it produces reasonable 8 to 12K programs and uses the MBI (subsidiary of the Royal Bank of Canada) MISAM multiple index sequential package via message handling with no adverse effects. However, when used to the limit, the dreaded 16K soon looms, and Airfix became paranoid about 16K RPL programs, until their efforts in BASIC+ let them to the same result!

In 1976, with an 11/70 gobbling up the work, the decision was made to write a MACRO version of RPL, with a variable length paged (virtual core) decision table structure and this came on stream in January 1977, after one years work. This development only became possible through the Oregon Museum of Science and Technology (OMSI) RT11 Run-Time System which enabled the production of an efficient compiler and runtime system under RSTS/E. The system has now achieved the ultimate accolade and the DEC Product Line have been unable to update Airfix with the source code to RSTS/E version 6B. The biggest problem resulting from this has been to fathom the 6B message passing interface which was needed to implement the MISAM functions in RPL. This and implementing MODE 1 facilities were cause for dectective work. In the absence of sources this was done by a combination of guesswork and a touch of BASIC.RTS decompilation.

The biggest task handled so far in RPL has been the re-write of an export order packing feedback program which has resulted in a 2400 line RPL program replacing four chained 16k BASIC+ programs and running three and one half times as quickly (wall clock) in 12k with considerable use of virtual memory - the development is awaited of a system optimiser to vary continuously individual programs' memory spaces with total machine load (RPL's 8K run time system allows 24K programs).

WRITING PROGRAMS IN RPL

There are three aspects to writing programs in RPL that are of interest to the general RSTS/E user:-

1. Staying compatible and co-existing with existing programs.

   Many RSTS/E installations have a variety of file structures dictated by the absence in early RSTS/E of variable length and indexed sequential file processing. As well as all the DEC Sort Compatible, Record I/O, and ASCII, "standard" file structures, RPL has a built in fund of "non-standard" indexed file structures that it recognises and in addition allows further options:- (a) to write in MACRO a file handler that is shareable and added to the run time system; (b) to write in RPL a file handler that is included automatically in user swappable code; (c) to write a file handler within the user code in an RPL program.

2. Extending the options open in RSTS/E to systems designers and programmers.

   Many RSTS/E systems are trapped between conflicting limitations such as:- the 64K block limit on file size; the difficulty of combining variable length records and multi-user file sharing; the difficulty of having chained record concepts and multi-user file sharing; the memory size trade offs between complicating file access and leaving room for application oriented code; the trade off between bad response times using index sequential techniques for all record seeking, and the multi-user access problems but faster responses of chain structured files.

   RPL has a built in chained file structure with variable length records that supports both multi-user access and secondary access by multi-index ISAM. Airfix now have in use a range of applications ranging from Sales Ledger (Receivables ?), through Order Processing, and on to Bill or Materials Processing running on 11/70, and 11/34 using this technique with multi-user access. All of these applications are complicated examples of their type and all are good on response times.

3. Generating Instant Systems

RPL offers the usual "program to write programs" option, but with a few interesting facilities:

- A program that asks questions about field definitions and sets up tables for a table driven validator (as distinct from normal RPL validation and prompts using DICTIONARY names).
- Automatic update or insertion of records into a chained file structure.
- User selected options to spread input across a line for speed or to drop into "idiot mode" and have each prompt given separately.
- A sensible interface to user code.

An example of RPL coding is given in the following text, and this is selected to demonstrate facilities and syntax, rather than to be a good example of programming.

```
*  EXAMPLE
*  LIST OF OUTSTANDING INVOICES PRIOR TO 31/12/76
*PFILE JIRPL4.OUT
*FILE  S  TRANS.SLD
*DICTIONARY
  TTYPE  =  0/2, ACCNO   =  4/4,  INVNO    =  8/6,  DATE  =  20/6
  YEAR   =  24/2, VALUE  = 26:8,  DISCOUNT = 42:8,  AREA  =  2/2
*INLIST
 M ACCNO,   N AREA,   1 VALUE,  2  DISCOUNT,  A DATE,  B  INVNO
*HEAD
  LISTING OF OUTSTANDING INVOICES PRIOR TO 31/12/76
  CUSTOMER   INVOICE NO.   DATE     VALUE     DISCOUNT
*OUT L  1,0
  NNMMMM     BBBBBB      AA/AA/AA  $1111.11  $222.22
*OUT M  1,0
 'CUSTOMER TOTAL'                  $111111.11  $222.22
*OUT N  1,1
 'AREA TOTAL'                     $1111111.11 $2222.22
*OUT F  1,0
 'FINAL TOTAL'                    $1111111.11 $2222.22
*DETAB RECORD
C TTYPE  =  '21'   Y   [SELECT TRANS TYPE 21(INVOICES)
  YEAR   LE '76'   Y   [WITH YEAR EARLIER THAN 1977
*GO
```

```
  LISTING OF OUTSTANDING INVOICES PRIOR TO 31/12/76
  CUSTOMER   INVOICE NO.   DATE     VALUE     DISCOUNT

  211144     005612      09/01/76   $590.00   $30.00
  211144     005694      16/01/76   $885.00   $45.00
  211144     005648      14/01/76   $472.00   $24.00
  211144     005695      12/01/76   $708.00   $36.00
  211144     005624      13/01/76   $708.00   $39.00
  211144     005696      13/01/76  $1062.00   $58.50
  211144     005649      15/01/76   $566.40   $31.20
  211144     005697      14/01/76   $849.60   $46.80
  CUSTOMER TOTAL                   $5841.00  $310.50
  211313     005605      05/01/76   $321.60   $12.20
  211313     005662      14/01/76   $482.40   $18.30
  211313     005632      13/01/76   $257.28    $9.76
  211313     005663      15/01/76   $385.92   $14.64
  379298     005622      16/01/76   $102.60    $0.00
  379298     005688      15/01/76   $153.90    $0.00
  379298     005645      16/01/76    $82.08    $0.00
  379298     005689      16/01/76   $123.12    $0.00
  CUSTOMER TOTAL                   $9217.89  $931.50
  AREA TOTAL                      $24073.43 $2070.00

  FINAL TOTAL                     $37668.31 $2735.09
```

DISCUSSION OF EXAMPLE

This example is a very simple listing program dealing with a sort compatible file of invoice details. The RPL statements are as follows:-

*PFILE JIRPL4.OUT specifies that the printfile is to be off lined onto a diskfile called "JIRPL4.OUT"

*FILE S TRANS.SLD specifies that a sort compatible file (type "S") called "TRANS.SLD" is to be read serially and automatically (record size obtained by RPL from the header block)

*DICTIONARY gives data names to fields in the record, defined by start position and length in bytes, where the symbol "/" means character and ":" means floating point. RPL also allows two word floating point (concurrent with four word) and very fast boundary aligned integer operations. Relative start positioning is allowed and recommended.

*INLIST associates characters from the alphabet with field names from the DICTIONARY, and these characters are later used to describe print lines for output. Some characters have extra significance:- "M", "N" etc. are control break fields in ascending order of significance; "1", "2" are fields to be totalled. Options are given to re-specify these characters if there are not enough of any one kind

*HEAD is followed by the page heading to be output at the top of each page. Page counting, date, time, channel throwing, before and after line skipping, can all be specified as part of the *HEAD statement and separate *HEAD's can be specified for each of different control break levels.

Each *OUT is followed by a picture of the print line(s) to be output at the control break level it represents (L for List or record level, M, N etc..) Each *OUT allows line or channel throwing and other controls and there can be many *OUT's for one control break level.

*DETAB RECORD is a decision table entered for each record of the *FILE and which has a "Y" for "YES" after each of its two conditions, that the transaction type (TTYPE) should be "21" and the year less than or equal to (LE) "76". This is a shorthand to indicate that all other conditions be ignored and could also be stated as:-

Condition 1	TTYPE = "21"	Y	ELSE
Condition 2	YEAR LE "76"	Y	-
Action 1	Accept Record	X	-
Action 2	Ignore Record	-	X

*GO instructs the RPL compiler to run the program after compilation.

The example is also coded in BASIC+ which shows that, though simple, it would be both easier to understand and to modify in RPL.

```
10    !  EXAMPLE
20    !  LIST OF OUTSTANDING INVOICES PRIOR TO 31/12/76
30    OPEN  "TRANS.SLD" FOR INPUT AS FILE 1%:
      OPEN  "TRANS.LST" FOR OUTPUT AS FILE 2%
40    ON ERROR GOTO 1000
45    B2$ = "000000": L% = -1%
50    DIM V(4),D(4)
52    GET  1%          !HEADER REC
53    FIELD  1%,2% AS B$,2% AS R$,2% AS S$     :
      B% = CVT$%(B$): R% = CVT$%(R$): S% = CVT$%(S$)
55    N% = 512/S%: T% = 1%            !N% RECS/BLK  T% CURRENT BLOCK NO
57    C% = 1%: GOTO 65
60    C% = 0%: GET  1%: T% = T%+1%    !C% POS'N IN BLOCK
65    IF T% = B% THEN N% = R%         ! LAST BLOCK,SET NO.RECS=NO.IN
BLOCK
70    IF C% = N% GOTO 60              !FINISHED WITH THIS BLOCK
73    FIELD  1%, C%*S% AS X$, 2% AS A$,6% AS B1$,6% AS D$,6% AS E$,
      2% AS F$,2% AS G$,2% AS H$,8% AS I$,8% AS K$,8% AS L$
75    V(0) = CVT$F(I$):  D(0) = CVT$F(L$): C% = C% + 1%
80    IF A$ <> "21" OR H$ > "76" GOTO 70              !SELECT RECORDS
90    IF B2$ = "000000" THEN LSET B2$ = B1$           !FIRST RECORD?
100   IF B2$ <> B1$ THEN GOSUB 300                   !CONTROL BREAK
110   GOSUB 700:PRINT  2% TAB(2); B1$; TAB(13); D$; TAB(25); F$;
"/";G$;
      "/"; H$; : I% = 0%: GOSUB 200
120   GOTO 70
200   PRINT  2% TAB(33);:
      PRINT  2% USING "$$      .    $$    .  ",V(I%),D(I%): L% = L%-1%
210   V(I%+1) = V(I%+1)+V(I%): D(I%+1) = D(I%+1)+D(I%)   !ROLL TOTALS
220   V(I%) = 0: D(I%) = 0: RETURN                       !ZEROISE TOTS
300   PRINT  2% "  CUSTOMER TOTAL"; : I%=1%: GOSUB 200
310   IF LEFT (B1$,2%) <> LEFT(B2$,2%) THEN PRINT  2%
      "  AREA TOTAL";: I% = 2%: GOSUB 200: L% = L%-2%: PRINT  2%
320   IF B1$ = "AAAAAA" THEN PRINT  2% "  FINAL TOTAL"; :
      I% = 3%: GOSUB 200
330   LSET B2$ = B1$: RETURN
700   IF L% < 0% THEN L% = 57%: PRINT  2% CHR$(12)
      "  LISTING OF OUTSTANDING INVOICES PRIOR TO 31/12/76":
      PRINT  2% "  CUSTOMER   INVOICE NO.   DATE     VALUE
DISCOUNT":
      PRINT  2%
710   RETURN
1000  LSET B1$ = "AAAAAA": GOSUB 300: CLOSE 1,2
1010  END
```

MORE COMPLICATED SYNTAX AND OPERATIONS

When the tabulator is in use access is allowed for own code inserts (subsidiary file look-ups etc.) at all significant points e.g.

- As every record from the driving file is read
- At the beginning and end of each control break (after the equal key compaction in *SORT)
- Before each print line is printed.

Programmers experienced with tabulators often spend time "bending" the system, and access at all these points facilitates this e.g. to convert totalled singles into dozens and singles at a control break total printing stage (which is an acid test of a tabulator).

Where the automatic tabulator is not used printing is performed by RPL through the association of a *PICTURE with a *LIST. A *PICTURE is a "picture" of a printline or group of lines e.g.

```
*PICTURE  DATA.LINE
AAAAA     BBBBBBBBBBBBBB      $1111.11   'INVOICE-NO'   CCCC
```

and a *LIST associates descriptor characters with *DICTIONARY field names.
e.g.

```
*LIST  DETAIL.FIELDS
A  ACCOUNT.NO,    B  ACCOUNT.SHORT.NAME,    1  TRANSACTION.VALUE,
C  INVOICE.NO,    D  TRADING.LIMIT,    F  REFERENCE
```

and the print verb asociates the *PICTURE "DATA.LINE" with a *LIST "DETAIL.FIELDS" to achieve printing e.g.

```
*DETAB PRINT.A.LINE
C TRANSACTION.VALUE  >  0                          Y   N
A ACCOUNT.SHORT.NAME  MV  'ERROR-*****'            -   X
  PRINT DATA.LINE  DETAIL.FIELDS                   X   X
```

Strings and variable length fields are handled with auto-incrementing pointers e.g. the following would be a loop:-

```
*DETAB POINTER.EXERCISE
C (POINTER)/3+  =  (POINTER)/-3          Y   N
A GOTO  POINTER.EXERCISE                 X   -
```

The following action indicates a variable length operation:-

```
A (POINTER)/(LENGTH) S [space fill a variable length field
```

or better:-

```
*DICTIONARY
VARIABLE.LENGTH.FIELD  =  (POINTER)/(LENGTH)
```

and:-

```
A  VARIABLE.LENGTH.FIELD  S
```

An initial action is allowed at the beginning of each decision table before conditions are evaluated, to allow setting of fields etc - the GOTO verb goes back to the beginning of the initial actions, while the REPEAT verb goes back to the beginning of the conditions.

Bit manipulations are achieved by direct addressing e.g.

```
C  44B?  =  "0"    0 1 2 3 4 5 6 7
A  CALL TABLE ?    A B C D E F G H
```

would test all eight bits of character position 44 and take different actions if these were set by calling appropriate DECISION TABLES called TABLEA through to TABLEH.

A two day course is sufficient for a programmer to learn RPL, with a one day follow-up to teach frills.

TECHNICAL DESCRIPTION

The RPL11 compiler produces a program consisting of a number of varaible length logical pages, each page corresponding to one logical unit of the program, e.g. a decision table. At run time, the pages are read into an area of store called the page pool, as and when required. When the page pool becomes full, those pages no longer required are removed, and the page pool is compacted to make room for the page requested at the time the page pool was full. Thus program size is only limited by the restrictions on file size imposed by the particular operating system, in the case of RSTS, 64K times 256 words = 16 megabytes.

Directives

RPL uses directives, which are source statement lines with an asterisk in the first position, to introduce the different logical units of an RPL program. Some directives expect detail lines following, a detail line being a source line without an asterisk in the first position. The primary directives in RPL are for introducing Decision Tables, Pictures, Lists, and files. The directives for these are:-

Directive	Page type
*FILE, *TABLE	1
*PICTURE	2
*LIST	3
*DETAB	4

Each occurrence of one of these directives creates a page of target code, and an entry in the page table. Pages are given unique names either explicitly by the programmer, e.g. *DETAB MAIN, *PICT PIC1, or implicitly by the compiler, e.g. in the case of *FILE, where asterisk followed by the channel number becomes the page name. The page name is used in the language as the means of linking pages together, e.g. CALL MAIN, PRINT PIC1 LIST3.

There are directives which do not generate pages of target code, e.g. *OPT, for specifying compiler and run-time options, and *DICTionary for defining fields and mapping data areas. Like the primary directives, these may or may not require detail lines following.

A third set of directives is reserved for use with the automatic reporting feature of RPL. These are *INList for specifying control break and totalling fields, and *TITLE, *HEAD, and *OUT for formatting report output. These directives are really built in macros in the language, since the compiler converts them into one or more of the primary directives. e.g. *INL generates three *DETABs and one *LIST.

I/.C RULE MASK TECHNIQUE

Decision Table Execution

Decision Tables in RPL are not pre-processed into a tree-structured branching program, as is usual with most decision table pre-processors. Instead the logic is directly executed by means of rule-masking, a technique first suggested by Kirk [2]. The rule masking method in RPL works by holding a mask with as many bits as there are rules in the table, with all bits initially set to 1 on entry to the table. As each condition is tested, those bits which correspond to rules which can no longer be satisfied having tested the condition are set to zero. When all conditions have been tested, the rule mask has 1 bits set in positions corresponding to satisfied rules, all other bits being zero.

At this point the rule mask may have none, one, or more than one bit set. If no bits are set, the ELSE rule is satisfied. If one or more bits are set, then the actions are executed by matching the rule mask with the relevance mask for each action in turn, and if any bits that are set to 1 match, the action is executed. It should be noted that if more than one bit matches for a particular action (the action has been found to be relevant for more than one rule), then that action is executed only once. For a fuller description of the technique, see [3].

Rule masking code

The rule mask in RPL11 is held on top of the processor stack, in (SP). It is pushed there by means of MOV £Rulemask,-(SP), on entry to a decision table, and popped off just before the end by TST (SP)+. This ensures that it is preserved on a CALL to a lower level decision table, and that it provides a true/false reply by setting the condition code on exit.

The decision table:-

```
C TYPE =  '?'   A  B
A WORK MV 0     X  -
  WORK + 1      -  X
```

generates the code:-

```
        MOV     £140000,-(SP)
        CMPB    TYPE,£'A
        BEQ     1$
        BIC     £100000,(SP)
1$:     CMPB    TYPE,£'B
        BEQ     2$
        BIC     £040000,(SP)
2$:     BIT     £100000,(SP)
        BEQ     3$
        CLR     WORK
3$:     BIT     £040000,(SP)
        BEQ     4$
        INC     WORK
```

```
4$:     TST     (SP)+
        RTS     PC
```

THE PAGING MECHANISM

The Page Table

The compiler builds a page table with a 4-word entry for each page, which looks like this at run-time:-

Word 0 Address of page in store if it is in store, else address of word 1 of this page table entry

Word 1 JSR R4,@-(R5). -(R5) is the address of the routine which gets a page

Word 2 Block no on disc program file, of start of page

Word 3 Length of page. Set negative if the page must be locked in store

Pages are numbered successively according to the order of mention of the pagename in the source program, whether it be a definition or use. This enables one pass incremental compilation. At run-time register R5 is permenently pointing at the start of the page table, and pages are called or entered by means of JSR @PageNumber(R5) or JMP @PageNumber(R5). If the page is in store it is entered immediately, otherwise it is fetched from disc, after making room for it if necessary, and then entered. We can't make Word 0 of the page table entry point directly to the page fetching routine, because there would be no way for the page fetching routine to tell which page to fetch !. That is the purpose of the JSR R4,@-(R5) in Word 1, which sets R4 to the address of the PTE for the page fetcher.

Page pool compaction

The method used for making room for a page when it wont fit in store is to throw everything out except for pages locked in by having word 3 of their page table entry set negative. Locked in pages are moved to the beginning of the page pool, and the space freed is left as a contiguous area at the end.

Any method of keeping a proportion of pages in store based on their frequency or recency of use suffers from the overhead of having to keep the frequency or recency statistics. The extra code needed for this on every call and exit increases the execution times of call and exit by a significant factor. Based on experience gained with a software paging system for a similar language on another machine (FTL6

on ICL1900), which does use a Least Recently Used algorithm for aging pages out of store, I felt that the simpler method adopted in RPL11 would be faster for the majority of programs written in the language.

TARGET CODE

The output of the compiler is referred to here as Target Code, to avoid confusing it with the Object Code output from most DEC compilers. The reason for this distinction is that Object Code needs to be linked by LINK or TKB to produce a runnable program. RPL11 Target Code on the other hand is directly executable and needs no linking.

At a very early stage in the design of RPL11 I had to decide between using interpretive code or directly executable machine instructions for the target of the compiler. I designed an interpreter and interpretive code for RPL11, which had the advantages of being small, typically needing only half the amount of store that PDP11 instructions need, and easier to compile, because of its closer relationship to the source language. The only disadvantage of this approach was that speed of execution of the interpretive code was at best twice as slow, and at worst five times slower than PDP11 machine instructions.

Since the slowness of BASIC+ was one of the major reasons for producing a competitor (the othar main reason being the 16k limit on BASIC+ programs - already catered for in RPL11 by paging), I finally and reluctantly decided to make the target code equal PDP11 machine instructions wherever possible, with calls to RTS service routines where necessary. The reason for this reluctance was that it meant more work in writing the compiler

Target Code/RTS linkage

One of the principal design aims for the Compiler/RTS was to allow enhancement and modification of the RTS without necessitating recompiling or re-linking existing RPL compiled programs. There are three methods already in use of linking programs to the services of a Run-Time System, on the PDP11:-

1) Use of the EMT and/or TRAP instruction. This satisfies the no re-compiling, no re-linking design aim, but is rather slow, especially under RSTS where there is memory management between the EMT in the object program, and the service routine in the RTS.

2) Using an interpreter with coded operation request codes, like the push-pop code of BASIC+. This approach was again rejected for speed reasons.

3) Using a linkage editor, such as TKB, to resolve addresses within the RTS. This does not satisfy the no re-linking design aim.

In the event, the technique used in RPL is a combination of 2 and 3. Entry to the RTS is made through JSR's and JMP's, but using indirect addresses. There is a dispatch table at the beginning of the RTS with word addresses of all the RTS service routines, and a register R0 is permenantly pointing to the beginning of this table. Entry to the RTS is made via JMP @Requestcode(R0) or JSR @Requestcode(R0), where Requestcode is the number of the service routine required. There are two exceptions for the most common entries to the RTS. These use the two short forms JSR (R0) and JSR @-(R0), to save on object code generated. This object code/Run time System linkage is similar to the page linking mechanism described earlier.

COMPILER DESIGN

The RPL11 Compiler consists of a root phase and six overlays. Each of the six overlays compiles one or more RPL11 directives of the same or similar type. The following table shows which overlays handle which RPL directives:-

Overlay no.	Directives handled
1	*FILE,*IFILE,*OFILE, *IOFILE,*PFILE,*TABLE
2	*PICT,*HEAD,*OUT
3	*LIST,*INL
4	*DETAB
5	*OPT,*MAX,*DICT
6	*STOP,*GO

The area of free space above the overlay area is used to hold the page of target code being compiled and the variable symbol table. Pages of target code are only written to disc when completely compiled, avoiding the need to read back any of the program into store while compiling.

The page table is built up one disc block at a time, and on becoming full is written to disc and chained to the next page table block. This chain of page table blocks is unwound at program load time to make the page table occupy a contiguous area of store.

The symbol table is made up of variable length entries, one for each pagename or data name. The symbol table entry holds a symbol type, making it possible to have a page name the same as a field name.

The following diagram shows the compiler storage map, followed by an exploded view of a symbol table entry.

ROOT + COMMON

OVERLAY
AREA

TARGET CODE
PAGE

FREE STORE

SYMBOL TABLE

Compiler Storage map

T D D D D S Y M B O L L

T = Symbol Type
D = Symbol definition.
varies from symbol
type to sybol type
L = Length of whole
symbol table entry

Symbol table entry format

RPL11 FILE HANDLING

RPL11 provides a powerful mechanism for accessing files structured in a variety of ways. A single letter on the file definition directive identifies the file to be handled by RPL11. The following standard types of file can be handled by file handlers built into the Run-Time System:-

A	-	Ascii
V	-	Virtual array
R	-	Record I/O
S	-	Sort compatible
D	-	Indexed chained variable length record
M	-	Misam indexed sort compatible

RPL11 Provides the following file manipulation verbs:-

C	LOOKUP	-	Positions file for next read via index if file type=D or M, else via record no
A	READ	-	Reads a record
	WRITE	-	Inserts a new record
	REWRITE	-	Overwrites an existing record
	KILL	-	Deletes a record
	NOTE	-	Takes a note of the

position of the latest record read/written

POINT - positions a file for next read without using index if indexed file.

Further file handlers may be written by the user to handle installation dependent file structures, and these may be incorporated into the run-time system or written as sharable file handler pages. If a new file handler is to be incorporated in the RTS it must be written in MACRO11 and conform to RPL file handler standards.

If it is written as a shareable file handler page it may be written in MACRO11 or RPL11, and put on a file called the library page file. The linkage of a shareable file handler page to a program requiring it takes place at run-time, thus obviating the need to recompile or re-link programs when changes are made to the file handlers.

RPL STRING HANDLING

RPL11 provides variable length string support through the use of indirect length field definitions. These take the form:-

Start position/(L) , where L must be a word aligned binary field of length 2 bytes. It is possible to make Start position indirect as well, through the use of a pointer thus:-

(P)/(L)

When INPUTing into a field defined with indirect length, the value of L is set to the number of characters actually typed, excluding the terminator.

RSTS/E Questionaire for Spring 1977 DECUS

The steering committee of the RSTS US SIG would like to have your comments on the RSTS portion of this Symposium. As the attendence of RSTS users grows the interest in sessions dealing with new and diverse uses of RSTS also grows. To keep up with your needs, we need your comments and ideas for furture meetings. Please fill out this questionaire and deposit it in the box marked RSTS SIG Questionaires.

.....

A. How do you rate this meeting?

1. List the sessions that were the most useful to you. Why?

---

2. List any sessions that did not live up to your expectations. Why?

---

3. What sessions need to be added at the next DECUS Symposium?

---

4. Were the handouts for the sessions helpful? How can they be improved?

****

B. Some questions about the RSTS SIG.

1. What new projects and what new working groups would you like to see started by the SIG?

---

2. What changes would you like to see in the newsletter?

---

3. How do you feel the DECUS library could be improved for you?

****

C. What kind of RSTS user are you:

1. How many years have you used RSTS? _____

2. How is your machine used? _____

3. What is you Job function on the machine? _____

4. How many DECUS Symposium have you attended previously? _____

****

Name and Address addtions or changes for the newsletter:

Add me to the newsletter mailing list ..... Correct my address .....

NAME .....

COMPANY .....

ADDRESS .....

ADDRESS .....

CITY ..... ST ..... ZIP .....